Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Návrh informačního systému v UML

Bc. Viacheslav Spitsyn

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Viacheslav Spitsyn

Systémové inženýrství a informatika
Informatika

Název práce

Návrh informačního systému v UML

Název anglicky

Design of Information System in UML

Cíle práce

Cílem diplomové práce je návrh informačního systému pomocí jazyka UML (Unified Modeling Language) na základě analýzy stávajícího systému, interní aplikace sloužící k zobrazení složek s dokumenty dostupných uživatelům. Nově modelovaný systém je navržen jako náhrada systému stávajícího za účelem odstranění kritických zranitelností systému, zlepšení jeho stávajícího stavu a přidání nových procesů.

Metodika

Diplomová práce bude obsahovat studie odborné literatury a internetových zdrojů ve formě literární rešerše. Dalším krokem je pomocí získaných znalosti zpracování vlastní práci, tj. analýza a následující projektování informačního systému pomocí jazyka UML. Bude provedena analýza stávajícího stavu systému, definovány kritické zranitelnosti, které by měly být odstraněny, následně budou sebrány funkční a nefunkční požadavky pomoci pohovorů s uživateli a stanoveny požadavky na novou funkcionalitu. Na základě získaných informací bude navržen nový systém v jazyce UML, a to pomocí diagramů tříd, stavových diagramů a diagramů interakcí. Podle dosažených teoretických a praktických znalostí budou formulovány závěry diplomové práce.

Doporučený rozsah práce

50 – 70

Klíčová slova

UML, informační systém, analýza, návrh, objektově orientované modelování

Doporučené zdroje informací

BLAHA, M. – RUMBAUGH, J. *Object-oriented modeling and design with UML.* Upper Saddle River, NJ: Pearson Education, 2005. ISBN 0130159204.

BRUCKNER, T. *Tvorba informačních systémů : principy, metodiky, architektury.* Praha: Grada, 2012. ISBN 978-80-247-4153-6.

FOWLER, M. *UML distilled : a brief guide to the standard object modeling language.* Boston: Addison-Wesley, 2004. ISBN 0321193687.

SCHMULLER, J. *Myslíme v jazyku UML : knihovna programátora.* Praha: Grada, 2001. ISBN 80-247-0029-8.

VRANA, I. – ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE. PROVOZNĚ EKONOMICKÁ FAKULTA, – ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE. KATEDRA INFORMAČNÍHO INŽENÝRSTVÍ. *Projektování informačních systémů s UML.* V Praze: Česká zemědělská univerzita, Provozně ekonomická fakulta, 2008. ISBN 978-80-213-1817-5.

Předběžný termín obhajoby

2021/22 LS – PEF

Vedoucí práce

Ing. Jakub Konopásek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 30. 11. 2022

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 30. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 11. 01. 2023

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Návrh informačního systému v UML" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30.03.2023 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Jakubovi Konopáskovi za odborné vedení, cenné rady a věnovaný čas při zpracování diplomové práce.

Návrh informačního systému v UML

Abstrakt

Cílem této diplomové práce je návrh informačního systému v UML. Pro dosažení cíle, budou představeny základní pojmy spojené s informačními systémy obecně, druhy přístupů pro analýzu a návrh daných systémů a také s modelovacím jazykem UML. Vlastní část práce bude věnována návrhu nového informačního systémů v UML, který by měl nahradit stávající interní bankovní systém pro prohlížení složek s dokumenty a jejich následnou správu uživateli. Důvody pro zavedení nového systémů jsou: potřeba odstranění kritických zranitelnosti, celková modernizace a zavedení nové funkcionality. Provedená analýza stávajícího systému a sebrané požadavky by mely sloužit základem pro vypracování návrhu IS v UML. V práci budou shrnuty dosažené výsledky a prostudovány možnosti realizace navrhovaného informačního systému.

Klíčová slova: informační systém, UML, analýza, návrh, objektově orientovaný přístup, diagram, životní cyklus IS, model, funkční a nefunkční požadavky, metodika

Design of Information System in UML

Abstract

The goal of this thesis is the design of an information system in UML. In order to achieve the goal, basic terms associated with information systems in general, types of approaches for analysis and design of these systems, as well as with the UML modeling language, will be introduced. The own part of the thesis will be devoted to the design of a new information system in UML, which should replace the existing internal banking system for viewing folders with documents and their subsequent management by the user. The reasons for introducing new system are: the need to remove critical vulnerabilities, overall modernization and the introduction of new functionality. The performed analysis of the existing system and the collected requirements should serve as a basis for developing an IS design in UML. The diploma thesis will summarize the achieved results and study the possibilities of implementing the proposed information system.

Keywords: information system, UML, analysis, design, object-oriented approach, diagram, life cycle of IS, model, functional and non-functional requirements, methodology

Obsah

1 Úvod

Návrh informačního systému je často kritickým aspektem úspěchu každé organizace. Efektivní IS musí splňovat potřeby svých uživatelů, poskytovat funkcionalitu potřebnou k podpoře cílů organizace a být dostatečně škálovatelný a flexibilní, aby se mohl přizpůsobit měnícím se požadavkům nebo se adoptovat na rychle se měnící svět informačních technologií.

Cílem této diplomové práce je návrh informačního systém, který vychází ze systému stávajícího, za účelem odstranění kritických zranitelností, zavedení nových funkcí a zlepšení celkové uživatelské zkušenosti. Systém představuje interní aplikace banky pro prohlížení složek s dokumenty a jejich následnou správu. Systém by měl poskytnout uživatelům bezpečné a intuitivní rozhraní pro přístup do složek a dokumentů. Proces návrhu začne analýzou stávajícího systému, po které bude následovat vytvoření nového návrhu, který zahrnuje nejlepší vlastnosti stávajícího systému a řeší jeho nedostatky.

Pro uvedení do problematiky zkoumané v rámci této diplomové práce bude samotnému návrhu informačního systému předcházet studium odborné literatury a webových zdrojů. Návrh informačního systému bude vytvořen pomocí modelovacího jazyka UML. UML (Unified Modeling Language) je široce používaný modelovací jazyk, který poskytuje výkonnou sadu nástrojů pro navrhování a dokumentaci informačních systémů. Samotný návrh bude tvořen pomocí tří modelů: modelu tříd, stavového modelu a modelu jednání. Model tříd poskytne pohled vysoké úrovni na součásti systému a jejich vztahy. Stavový model bude použit k modelování chování objektu v reakci na vnitřní a vnější události, tedy představuje různé stavy objektu a způsob přechodu mezi těmito stavy. Model jednání bude sloužit k popisu funkčních požadavků na systém z pohledu jeho uživatelů. Jako prostředky pro návrh informačního systému v rámci dané diplomové práci budou použity vybrané diagramy UML, které budou prostudovány v kapitole „Teoretická východiska“.

Tato diplomová práce bude demonstrovat důležitost a užitečnost použití UML při návrhu informačních systémů a poskytne základní představení tohoto procesu. Výsledný návrh by měl poskytnout pevný základ pro vývoj kvalitního informačního systému, který

odpovídá potřebám jeho uživatelů, splňuje všechny požadavky kladené na systém a podporuje cíle organizace.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem diplomové práce je navrhnout informační systém pomocí modelovacího jazyka UML. Navrhovaný systém by měl nahradit stávající systém interní bankovní aplikace, která umožňuje uživatelům prohlížení složek a správu dokumentů v nich umístěných. Návrhu nového systému bude předcházet analýza systému stávajícího.

Nový systém se navrhuje za účelem modernizace systému stávajícího, konkrétně k odstranění kritických zranitelností, přidání nové funkcionality, zlepšení uživatelské zkušenosti a přizpůsobení k současným bankovním standardům jak z hlediska bezpečnosti, tak i designu.

2.2 Metodika

Metodika diplomové práce je založena na studiu odborné literatury a webových zdrojů představeného formou literární rešerše. Bude prostudována problematika informačních systémů obecně a rovněž budou popsány důležité pojmy s tím spojené. Následně budou popsány druhy přístupů k analýze a návrhu informačních systémů i nástrojů CASE, které slouží pro podporu jejich zpracování. Poslední kapitola v rámci teoretických východisek bude věnována modelovacímu jazyku UML. Bude provedeno studium aspektů, tedy obecný popis, historie a ukázka vybraných nástrojů tohoto jazyka důležitých pro zpracování návrhu v části praktické.

Vlastní část práce bude věnována návrhu informačního systému v UML. Před návrhem bude analýza systému stávajícího, což poslouží pro pochopení důvodů zavedení systému nového. Dále pomocí rozhovorů s uživateli, vedoucím programátorem a vedením týmu budou sebrány funkční a nefunkční požadavky na navrhovaný IS. Následně na základě provedené analýzy a sebraných požadavků bude zpracován návrh informačního systému v modelovacím jazyce UML.

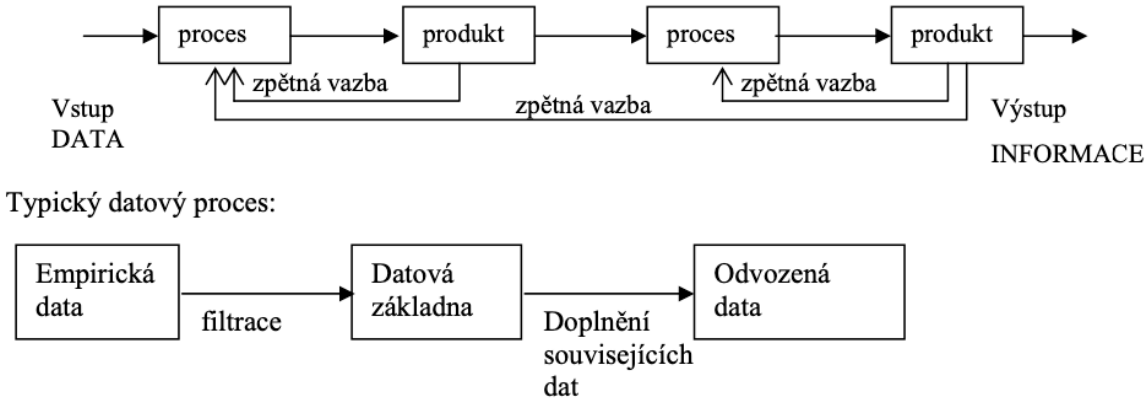
3 Teoretická východiska

3.1 Informační systém

Před ponořením do problematiky informačních systémů je důležité definovat pojmy s tím spojené. Jedná se především o data, informace a znalosti a jak mezi sebou souvisí.

Data představují vjemy, které popisují realitu. Data můžou být různých forem, například text, zvuk, obraz a fakta. Rozdělují se tyto vlastnosti dat: [1]

- nezávisí na konkrétním člověku, často udávají současný stav reality;
- neustále snižují složitost reality;
- i přestože tvoří významnou velikost, mnohdy mají hodně detailů;
- relativně často a rychle se mění;
- verifikovatelnost dat je možnost znovu objektivně zkontrolovat přesnost a správnost dat.

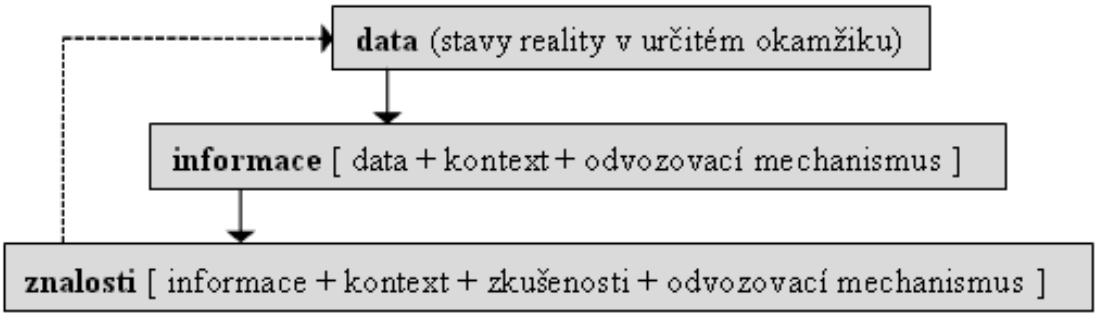


Obrázek 1 – Schéma typického datového procesu (Zdroj: [1])

Pomocí vhodného zpracování z dat vznikají informace. Dále je uvedena definice informace: „*Informace (information) je význam dat, jak je má chápat člověk. Data jsou fakta, informacemi se stávají tehdy, když jsou v kontextu a nesou význam pochopitelný lidem. Počítače zpracovávají data bez toho, že by rozuměly, co znamenají.*“ [2]

Znalost se formuje informací v kontextu. Ta představuje chápání dosažené zkušeností nebo studiem, je pochopitelná a uplatnitelná k řešení problémů či rozhodování. [3]

Data, informace a znalosti, nehledě na to, že úzce souvisí, jsou odlišné pojmy. Následující obrázek udává vztah a obsah mezi daty, informacemi a znalostmi:



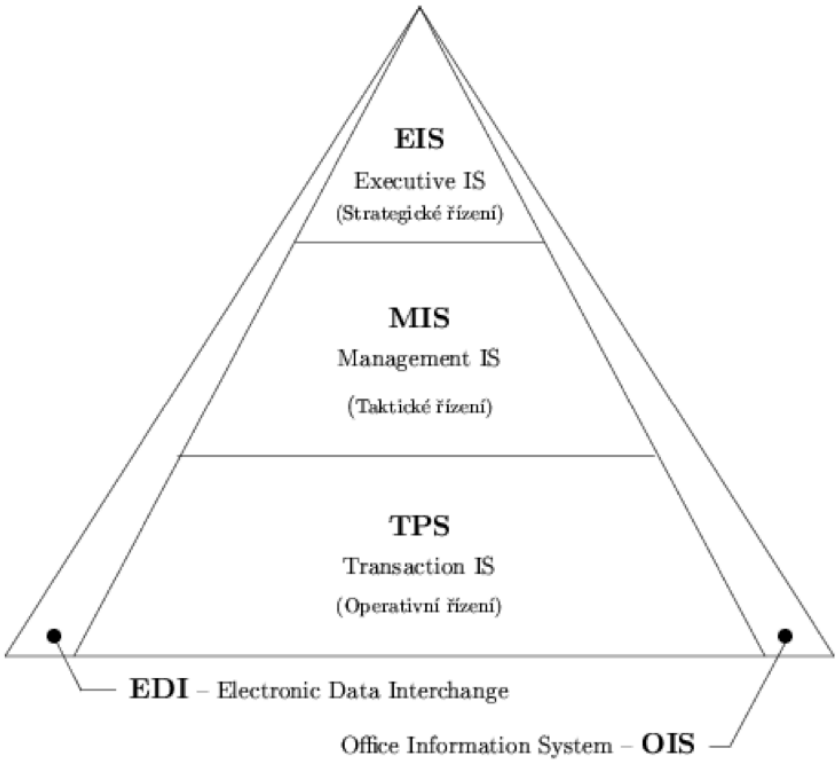
Obrázek 2 – Vztahy mezi daty, informacemi a znalostmi (Zdroj: [4])

Pojem informační systém má hodně definic. Na začátku lze uvést, co znamená obecně systém. Obvykle pod tímto pojmem se rozumí uspořádaná množina prvků a vazeb mezi nimi. Systém se vyznačuje vazbami jdoucími na vstup a výstup, pomocí kterých dostává informace z okolního světa a jiné informace do okolí vysílá. [5]

Dnes málokterý podnik se obejde bez informačního systému. Hlavním důvodem je, že pracuje s informacemi, kterých je čím dál tím více. Cílem informačního systému je zabezpečení správných informací na správném místě, a to ve správný čas. Jednou z definic je: „*Informační systém je soubor lidí, technických prostředku a metod (programů), zabezpečujících sběr, přenos, zpracování, uchování dat, za účelem prezentace informací pro potřeby uživatelů činných v systémech řízení.*“ [6] [7]

3.1.1 Klasifikace informačních systémů

Existuje několik způsobů, jak klasifikovat vývoj informačních systémů. Jedním z důležitých a nejznámějších, z důvodu že informační systémy slouží jako podpora pro řízení, je dělení podle řídící úrovně. Obrázek uvedený níže odráží danou klasifikaci a jmenuje se informační pyramida. [2] [8]



Obrázek 3 – Klasifikace informačních systémů dle úrovně řízení (Zdroj: [8])

Systémy datových transakcí nebo informační systémy pro transakční zpracování (TPS) – tyto systémy mají za předchůdce klasické dávkové systémy (např. agendové zpracování mezd nebo faktur). Cílem těchto IS je podpora hlavních činností podniku na operativní úrovni řízení. Nejistota zpracovávaných informací je velice nízká. Slouží především pro snadnější operace s daty a převážně se jedná o interní informace. Důležitou činností je zpracování dat a znalostí pro úrovně vyššího řízení. [2] [8] [9]

Manažerské informační systémy nebo systémy pro podporu rozhodování (MIS) – systémy podporující taktickou úroveň řízení. Pomáhají ke kvalitnímu a efektivnímu řešení úloh spojených se službami pro zákazníky, například to jsou výpočty cen. Zde se zvyšují požadavky na externí zdroje informací. [2] [8] [9]

Strategické informační systémy (EIS) – systémy jsou taktéž označovány jako BIS (Business Inteligence Systems). Dané systémy podporují strategickou úroveň a jedná se především o informace z okolí. Cílem EIS je posílení podniku na trhu. Zde se vytvářejí datové sklady nebo se používají analytické nástroje pro analýzu vzájemných závislostí. [2] [8] [10]

Také dané rozdělení IS obsahuje dva subsystémy: OIS (systémy pro podporu kancelářských činnosti) a EDI (elektronická výměna dat). Obojí tyto subsystémy lze nalézt na všech úrovních řízení. OIS zahrnuje standardní kancelářské a komunikační prostředky pro podporu kancelářských práci, např. pošta, různé messengery. EDI je standard pro elektronickou výměnu dat. Hlavni myšlenkou využití EDI je náhrada papírové dokumentace elektronickou, za účelem snížit náklady s tím spojené, zvýšit kvalitu a efektivitu a můžou být propojené různé IS jak uvnitř, tak i vně společnosti. [8] [11]

3.2 Životní cyklus informačního systému

Informační systém během své existence, od vzniku až po vyřazení z provozu, prochází řadou výrobních fází. Zde se jedná o tzv. životní cyklus a fáze životního cyklu informačního systému. Cyklus zároveň ukazuje pravděpodobný návrat k předchozím etapám. Existuje několik metodik, z nichž každá svým způsobem definuje a popisuje obsah vývojové a provozní fáze daného informačního systému. Metodiky založené na různých kritériích mohou vymezovat fáze životního cyklu informačního systému různými způsoby. Každá metodika se však vyznačuje rozdělením životního cyklu informačního systému na etapy, popisem obsahu každé etapy a zaměřením na určité etapy. [12]

Jako příklad lze uvést metodiku MDIS (Multidimensional Development of Information System), česky multidimenzionální vývoj informačního systému, který člení životní cyklus informačního systému do těchto fází: [12]

- IST – fáze informační strategie podniku,
- ÚST – fáze úvodní studie,
- GAN – fáze globální analýzy a návrhu IS,
- DAN – fáze detailní analýzy a návrhu IS,
- IMP – etapa implementace IS,
- ZAV – etapa zavádění IS do provozu,
- PÚR – etapa provozu, údržby a dalšího rozvoje IS.

Existují i metodiky, které by obsahovaly méně etap, rozdělení by vypadalo následovně: příprava IS, zavádění IS a provoz IS. Ale většinou se v dnešní době pracuje s více detailním rozdělením na větší množství jednodušších etap, například: [13] [14]

- Plánování a identifikace problémů – počáteční a velice podstatná etapa ve vývoji IS. Udává velký vliv pro dosažení úspěchu v realizaci celého projektu. V dané etapě se hledají konkrétní požadavky, které by měl splňovat nový systém. Hlavními činnostmi jsou: stanovení požadavků, charakteristika očekáváných výsledků, posouzení časové a technické složitosti, zvážení dostupnosti zdrojů potřebných pro realizaci, stanovení úrovně finančních prostředků. Jako první je potřeba provést hluboké plánování každé časti projektu a hlavním výstupem této etapy je detailní harmonogram. Taktéž během plánovací fáze je důležité zajistit prostředky a procedury sloužící pro řízení a kontrolu projektu.
- Analýza – zahajovací etapa zpracování IS. Jako výsledek, analýza by musela uvést nutné informace o společnosti, kde bude nový IS provozovat, o všech detailech tohoto systému a o tom, jak by měl fungovat. O výsledek dané fáze se stará analytik, který by se měl zabývat studiem uvnitř podniku a dokumentů obdržených od společnosti. K tomu má velkou řadu různých nástrojů, technik a metod.
- Návrh systému – cílem této fáze je označit, jak bude systém postaven, jaké bude obsahovat části a funkce. Zpravidla návrh systému je členěn do dvou částí na logický a fyzický návrh IS. Logický návrh má za úkol popsat funkcionalitu a fyzický komponenty, které jsou potřebné pro zabezpečení funckionality. Základním pramenem pro realizaci systému a zároveň klíčovým dokumentem v rámci dané fáze je podrobný návrh systému, obsahující i celou projektovou dokumentaci.
- Tvorba systému – v dané etapě vzniká informační systém. Za tvorbu odpovídají vývojáři databáze a zdrojového kódu. Jako další výsledek je kompletní dokumentace, která je vytvořena spolu s koncovými uživateli daného IS a která také zahrnuje různé návody a odpovědi na často kladené otázky atd. V rámci tvorby informačního systému probíhá testování jak celku, tak i jednotlivých funkčních částí.
- Implementace – tato etapa označuje uvedení systému do provozu. Prosperita dané etapy závisí na výsledcích všech předchozích fází.

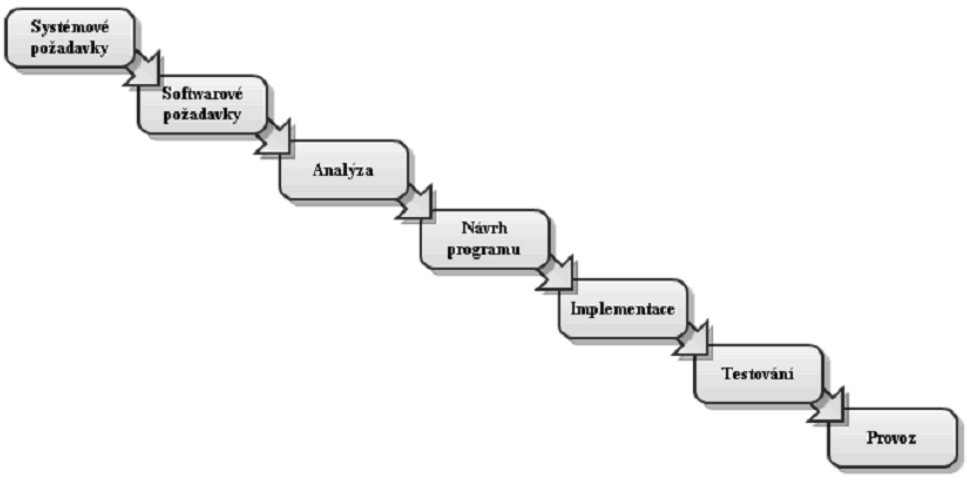
- Provoz a údržba – poslední etapa životního cyklu IS. Nejdůležitějšími aktivitami jsou opravy možných chyb, obnovení a zálohování dat, možné úpravy IS a podpora uživatelů. Úkolem v dané fázi je zabezpečení plynulého provozu a rozvíjení podle potřeb koncových uživatelů IS.

3.3 Modely ve vývoji životního cyklu informačního systému

Výběr vhodného modelu životního cyklu je klíčovým faktorem pro úspěch projektu. Modely životního cyklu IS udávají, jak mezi sebou souvisí etapy tohoto cyklu. Původními modely, ze kterých vychází velké množství současných modelů, jsou vodopádové a spirálové (iterativní a inkrementální) modely. [7]

3.3.1 Vodopádový model

Vodopádový model je nejstarším známým modelem životního cyklu systému. Tento model byl definován doktorem Winstonem W. Roycem v roce 1970 v článku nazvaném „Managing the Development of Large Software Systems“ jako podpora při rostoucí složitosti systémů v leteckém průmyslu. Jak ukazuje další obrázek, název tohoto modelu vyplývá z podoby procházením každé etapy protékaní vody vodopádem a obsahuje 7 hlavních etap: [15]



Obrázek 4 – Vodopádový model životního cyklu softwaru (Zdroj: [15])

Vodopádový model je založen na postupném přechodu od jedné fáze do druhé, tedy do další etapy vstoupí pouze, když předchozí byla již ukončena. Tento model není složitý

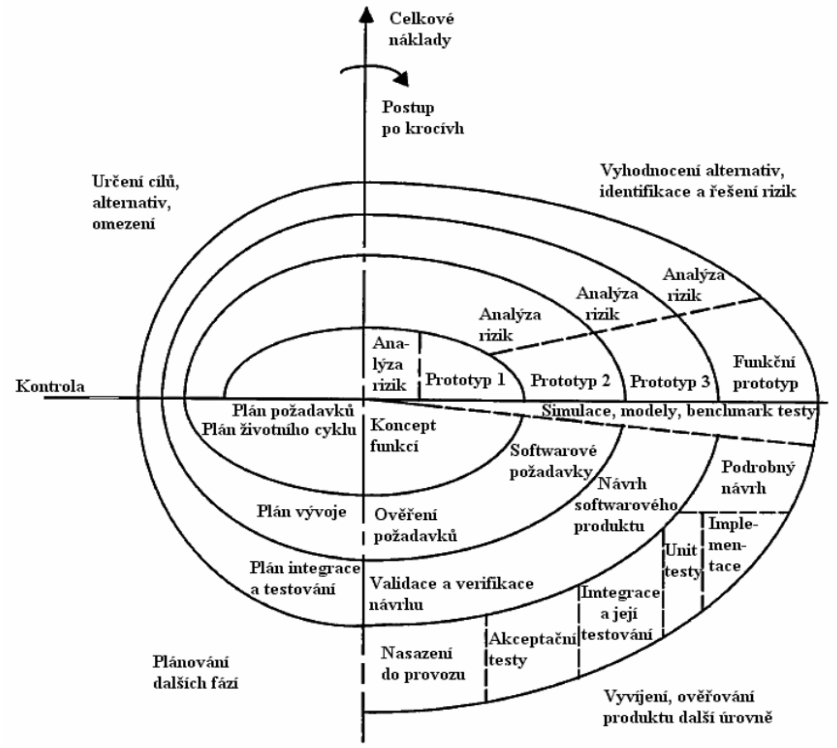
a je snadné ho aplikovat. Většinou se používá u projektů malého rozsahu, u kterých se nepředpokládá přidání nebo změny funkcionality. Úspěšnost celkového využití daného modelu záleží na tom, aby bylo zajištěno dostatečné množství času etapám předchozím. Jedině takhle lze docílit větších úspor během fází následujících. V porovnání z pozdějšími etapami (testování nebo provoz), stanovení a opravení chyb ve fázích na začátku (analýza) značně levnější.

Mezi hlavní nevýhody využití vodopádového modelu naleží, většinou u velkých projektů, že se nelze vrátit k jakékoliv předchozí etapě během vývoje, což neumožňuje rychle reagovat na měnící se požadavky uživatelů. Z pohledu testování, které nastává po implementaci, může nastat situace, kdy systém je skoro zpracovaný pro předání zákazníkovi, ale vyskytne se chyba, která vede k přepracování celého projektu.

Dnes je z důvodu jeho srozumitelnosti a jednoduchosti vodopádový model hlavně využíván pro výukové účely. Z tohoto modelu bylo vyvinuto množství modifikací, které se snaží zbavit nedostatku modelu původního. [16]

3.3.2 Spirální model

Spirálový model byl poprvé definován Barry Boehmem v roce 1986. Daný model slouží jako zlepšení modelu vodopádového a snaží se odstranit jeho hlavní nevýhody. Následující obrázek udává, jak vypadá spirálový model v publikaci Boehma: [17]



Obrázek 5 – Spirálový model životního cyklu IS (Zdroj: [17])

Spirálový model je založen na následujícím principu: přechod do další fáze závisí na důkladné analýze všech rizik a možných problémů, je tedy snazší se přizpůsobit změnám požadavků v průběhu práce na projektu, proto je pro velké projekty vhodnější než vodopádový model. Spirálový model se skládá z několika fází, které se neustále opakují až do dokončení projektu, dokonce i specifikace na začátku implementace je pouze hrubým popisem uživatelských požadavků a později se každou fází zpřesňuje. Životní cyklus podle spirálového modelu lze rozdělit do čtyř hlavních fází: [17]

- definování cílů, alternativ a omezení;
- vyhodnocení variant, stanovení a eliminace rizik;
- vývoj a ověřování produktu na další úrovně;
- plánování dalších kroků.

Obvykle se po každé fázi provádí testování, což je velkým přínosem, protože již na začátku se lze zbavit nežádoucího chování. [18]

3.3.3 Agilní metodiky

Agilní metodiky jsou náhradou metodik tradičních, které byly definovány předtím. Pomocí nich lze dodávat IS velmi rychle a adaptovat výsledné řešení v souladu s měnícími se požadavky odběratelů. Jde o množství metodik vzniklých v 90. letech 20. století založených na konceptu, co nejrychleji ověřit návrh systému, zpracovat ho celý nebo jenom část, ukázat klientovi, získat zpětnou vazbu a podle ní upravit řešení. Všechny metodiky se mezi sebou liší, avšak jsou založené na stejných myšlenkách. Existuje speciální dokument vyvinutý představiteli různých agilních metodik. Dokument definuje 10 nejdůležitějších aspektů agilní metodologie, které vypadají následovně: [19]

- hlavním cílem je včasné a nepřetržité dodávání produktu, přinášející určitou hodnotu klientům;
- požadavky lze měnit i v pozdějších fázích vývoje;
- každý den tým vývojářů a klienti vzájemně spolupracují na projektu;
- motivovaní jedinci, kteří mají vytvořeny podmínky pro práci a mají podporu vedení, jsou klíčovým faktorem úspěchu projektu;
- osobní komunikace je nejúčinnějším zdrojem pro síření informace mezi skupinou vývojářů;
- procesy v agilním vývoji očekávají tzv. zdravý vývoj;
- ideální technické zpracování a návrh;
- zásadním požadavkem je jednoduchost řešení, tj. umění minimalizovat množství neudělané práce;
- týmy, které jsou schopné se sami organizovat, dodávají nejlepší návrhy a architektury.

Mezi nejznámější metody agilního vývoje naleží například Scrum, extrémní programování, Lean Development, Feature-Driven Development (FDD).

3.3.3.1 Scrum

Scrum je zformován na ideji, že proces zpracování IT produktu není pevně definován, jak to mají tradiční metodiky, a kvůli tomu předpokládá absolutně jiný styl řízení. Název Scrum pochází ze sportovní hry rugby a lze ho přeložit jako mlýn, protože naznačuje týmovou práci. Pro Scrum je typická práce podle určitých časových horizontů. Ty se

nazývají Sprint, většinou 30denní, nebo 14denní, na konci kterých jsou představeny předem stanovené cíle. Jednou ze zásadních vlastností Scrum je provedení takzvaných Scrum Meetings neboli Daily, každodenních krátkých jednaní pro koordinaci mezi členy týmu. Každý má na starosti seznámit ostatní s tím, co řešil po předchozím meetingu, co bude řešit dnes a s jakými problémy se setkal. Podle metodiky Scrum se životní cyklus IS dělí na tyto 4 etapy: [19]

- plánování – specifikace funkčních požadavků;
- inscenování – přidání nefunkčních požadavků;
- vývoj – jedna, nebo více skupin doručuje určité funkce, které mají nejvyšší prioritu v rámci Sprintu;
- dodání – představení díla zákazníkům.

3.3.3.2 Extrémní programování

Metodika extrémního programování je známá jako „jednoduchá“, i když je to ve skutečnosti velmi disciplinovaný proces. Neexistuje žádný podrobný popis této metodiky, je založena na myšlence použití užitečných tradičních přístupů a postupů při vývoji a jejich povýšení na novou „extrémní“ úroveň. Hlavní techniky používané v extrémním programování jsou popsány níže: [20]

- párové programování – technika, která předpokládá práci dvou vývojářů na jednom počítači. Jeden z nich pracuje přímo s kódem programu, druhý se dívá na jeho práci a sleduje celkový obraz toho, co se děje. V případě potřeby se klávesnice volně předá druhému. Při práci na projektu nejsou dvojice pevně dané: doporučuje se je promíchat, aby každý programátor v týmu měl dobrou představu o fungování celého systému. Párové programování tedy zlepšuje interakce v týmu;
- testování jednotek a funkční testování – testování jednotek (unit-testy) umožňuje vývojářům se ujistit, že každá jednotka samostatně funguje správně. Taktéž pomáhá jiným developerům porozumět, proč je tato konkrétní část kódu podstatná a jak funguje. Během studie zdrojového kódu jednotky se její logika stává srozumitelnou, protože je vidět, jak by měla být použita. Funkční testování slouží pro testování funkčnosti logiky několika propojených částí systému. Jsou méně detailnější v porovnání s testováním

jednotek, ale pokrývají větší objem kódu, což vede k větší šanci odhalení chyb. Kvůli zmíněným vlastnostem funkční testování většinou má vyšší prioritu než testování jednotek;

- refaktorizace – je metodika zlepšení kvality kódu, a to beze změny jakékoliv funkcionality. Extrémní programování bere v potaz, že kód, který byl jednou napsán, se bude během práce na projektu neustále měnit, aby byla zajištěna, co nejlepší kvalita;
- to nejjednodušší, co ještě může fungovat – extrémní programování vychází z toho, že v průběhu práce nad projektem se požadavky zákazníků mohou opakovaně měnit, a tím pádem výsledný produkt by neměl být předem celkově navržen. Snaha navrhnout co nejdetailnější systém na začátku je ztráta času. Extrémní programování představuje návrh jako tak důležitý proces, že by měl být prováděn nepřetržitě po celou dobu životností projektu. Návrh se musí zpracovávat po malých krocích s ohledem na neustále se měnící požadavky. V každém okamžiku musí být použito co nejjednodušší řešení, které bude fungovat a pak je potřeba ho změnit, jakmile se změní požadavky;
- metafora – je analogem toho, co jiné metodiky nazývají architekturou. Metafora systému udává týmu přehled o tom, jak aktuálně funguje, kam se přidávají nové komponenty a jak by měly vypadat;
- nepřetržitá integrace – v extrémním programování integrace kódu celého systému probíhá několikrát denně, až jsou si vývojáři, že všechny testy jednotek fungují správně. Pokud integrace bude probíhat nepřetržitě, je možné se tím zbavit většiny problémů spojených s vývojem systému;
- plánovací hra – hlavním cílem plánovací hry je rychle představit hrubý odhad rozsahu práce a postupně ho měnit, až se upřesní požadavky.

3.3.3.3 Lean development

Metodika Lean development pochází z technik výroby a založena na tzv. konceptu dynamické stability, což znamená propojení schopnosti se přizpůsobit měnícím požadavkům se schopností vyrábět stabilně a pořád se zlepšujícími se vnitřními procesy. Hlavní myšlenkou Lean development je, že zlepšování efektivity a odstranění odpadu, tedy všeho,

co nepřináší nějakou výhodu zákazníkům, může být řízeno na všech úrovních projektu. Dále je popsáno 10 pravidel štíhlé výroby, které se aplikuji ve vývoji IS: [19]

- odstraňování zbytků – odstranit vše, co není podstatné pro konečný produkt;
- minimalizace zásob – ve vývoji se zásobami rozumí dokumentace. Jedním ze způsobů, jak zmenšit dokumentaci, je zvýšit úroveň abstrakce neboli méně detailů;
- minimalizace času vývoje – zmenšit práci na procesech, aby se celkový čas vývoje se zkrátil;
- vývoj tažený poptávkou – vývoj se musí přizpůsobit měnícím se požadavkům zákazníků;
- pracovníci s rozhodovací pravomocí – developeři musí rozumět tomu, jak jejich výkon pomáhá celému projektu a všem potřebným detailům, důležitým pro jeho dokončení;
- uspokojovat požadavky klientů – zákazníci nejsou schopni na začátku stanovit celou funkcionalitu. Proto je potřeba s tím počítat a neztrácet hodně času na úplnou specifikaci;
- zavedení zpětné vazby – tím se rozumí komunikace se zákazníkem pro zpřesnění a doplnění požadavků během vývoje;
- odstranění lokální optimalizace – optimalizace současného projektu nedává smysl v rychle se měnícím prostředí;
- partnerství s dodavateli – koupě nějaké součásti není špatný nápad, jelikož pro klienta je důležitý výsledek;
- zavedení kultury pro neustálé zlepšování – musí být prostředí a motivace pro zlepšení procesů během vývoje.

3.3.3.4 Feature-Driven Development (FDD)

Z angličtiny lze název Feature-Driven Development přeložit jako vývoj řízený užitnými vlastnostmi. FDD představuje snahu propojit nejznámější metodiky ve vývoji softwaru, které berou za důležitou pro zákazníka funkcionalitu. Hlavním cílem FDD je vývoj fungujícího softwaru systematicky v předem definovaný čas. FDD zahrnuje pět následujících kroků: [21]

- vývoj obecného modelu – vývoj začíná z celkové analýzy. Pak pro každou část budoucího systému se provádí detailnější analýza, pak se formulují modely. Propojením určitých modelů vzniká jeden model pro konkrétní oblast. Modely každé oblasti se dávají dohromady do obecného modelu, který se může v průběhu vývoje měnit;
- sestavení seznamu požadovaných funkcí – informace získané v průběhu vývoje obecného modelu se používají pro sestavení seznamu funkcí. Oblasti se dělí na podoblasti podle jejich funkcionality. Každá podoblast odpovídá jednomu procesu, jehož kroky se stávají seznamem funkcí neboli vlastnostmi, které pak budou součástí celkového seznamu;
- plánování rozsahu práce pro každou funkci – po sestavení seznamu požadovaných funkcí přichází plánování vývoje;
- návrh funkce – pro každou vlastnost se tvoří návrhový balík. Vedoucí programátor přikládá malou skupinu vlastností, které by měly být zpracovány během následujících dvou týdnů. Taktéž se zpřesňuje celkový model a probíhá kontrola návrhu;
- implementace funkcí – po úspěšné kontrole návrhu probíhá implementace funkcionality. Po provedení testování a ověření kódu, kdy se vyvinutá funkcionalita stává součástí výsledného systému.

3.4 Druhy přístupů pro analýzu a návrh informačních systémů

3.4.1 Strukturovaný přístup

Strukturovaný přístup pochází z oblasti programování softwaru v 60. letech 20. století. Jedná se o tzv. strukturované programování, jehož principy se dále postupně zobecňovaly a pak pronikaly i do oblasti analýzy a návrhu informačních systémů. Hlavními myšlenky tohoto programování jsou: [12]

- řešení se navrhuje shora dolů;
- systém se člení do modulů, sestavuje se hierarchie modulů;
- programuje se bez použití příkazu skoku.

Strukturovaný přístup se od 80. let 20. století dělí na funkční a datově orientovaný přístup: [12]

- funkční přístup – vychází z modelování systému jako souboru funkcí, které představují množinu vlastností měnících vstupy na určité výstupy. Funkce, přitom mezi sebou spojené datovými toky, což znamená, že výstupy jedné funkce přichází na vstup do další funkci. Hlavním modelem pro návrh IS je diagram datových toků (DFD). Podstatným je také datový model (ERD), který ukazuje strukturu datových uložišť;
- datově orientovaný přístup – se zakládá na struktuře a vlastnostech datových uložišť. V tomto případě hlavním nástrojem je datový model a diagram datových toků se odvozuje na základě nalezených procesů, které pracují s daty datového modelu.

Základní principy strukturovaného přístupu byly předloženy Tomem DeMarco v jeho práci „Strukturovaná analýza a specifikace systému“, ty vypadají následovně: [12]

- rozdělit systém na subsystémy,
- využívat grafické znázornění systému,
- logický model systému musí být před implementací.

Modelování systému strukturovaným přístupem se provádí na zakladě několika grafických nástrojů uvedených níže: [12]

- ERD (Entity Relationship Diagram) – pomocí diagramů datových modelů lze modelovat strukturu IS a datových uložišť. ERD slouží pro ukázku toho, jak jsou data uložena do datových uložišť a vzájemné vztahy mezi nimi. Datový model odráží statický pohled na systém;
- DFD (Data Flow Diagram) – diagram datových toků ukazuje datové nebo řídící toky v systému a popisuje funkce tohoto systému a odráží dynamický pohled;
- FSD (Function State Diagram) – cílem diagramu funkční struktury je ukázat rozdělení systému na subsystémy, zaznamenává funkční hierarchii IS a předkládá pohled na systém při nahlížení na jeho hierarchickou strukturu;
- STD (State Transition Diagram) – stavový diagram modeluje časově závislé chování systému a zahrnuje sled stavů, ve kterých část, nebo celý systém se můžou objevovat, a za jakých podmínek se muže ten stav změnit;

- Data dictionary – pomocí datového slovníku lze formalizovaně popisovat data systému z pohledu uživatele. Datový slovník může pomáhat k úpravě datového modelu a snížení redundance informačního nadbytku tedy duplikování dat v databázi;
- Structure chart – diagram struktury programového systému představuje hierarchie programových modulů IS ve formě grafu. Uzly ukazují dílčí moduly, když hlavní programový model představuje kořen;
- Flow chart – vývojový diagram má nejmenší důležitost pro modelování IS. Ukazuje algoritmus a používá se pro popis funkcí.

3.4.2 Objektově orientovaný přístup

Objektově orientovaný přístup pochází z 80. let 20. století a navazuje na strukturovaný přístup. Když se jedná o objektově orientovaném přístupu (OOP), hovoří se o objektech, na kterých je založen. Objekt představuje strukturu, která má vlastnosti neboli atributy, a chování, tedy množinu funkcí, které muže provádět. IS z hlediska OOP představuje množinu spolupracujících objektů. Základní pojmy OOP tvoří: [12]

- objekt – je prvek IS, skládá se z atributů a metod;
- třída – je skupina s podobnými vlastnostmi a chováním objektů;
- atribut – je část popisující vlastnosti objektu;
- metoda – je funkce, která ukazuje reakci objektu na nějaký jev.

Mezí nejdůležitější principy OOP patří: [23]

- identita – ukazuje, že každý objekt je jedinečný. I když můžou objekty nabývat stejných hodnot atributy jsou jiné;
- zapouzdření – znamená skrytí realizace metod a určitých hodnot atributů. Tedy podstatné je, co objekt dělá, ale ne konkrétní funkce probíhající pro dosažení tohoto chování;
- klasifikace – znamená to, že objekty tvořené stejnými atributy a metody seskupují do tříd. Třída je abstrakce, která popisuje chování podstatné pro daný konkrétní systém. Objekt je instancí třídy, která jich může mít nekonečně mnoho;

- dědičnost – udává schopnost objektu dědit vlastnosti předchozích objektů, předků, které mají logickou souvislost, což je děláno za účelem rozšíření svých vlastností;
- polymorfismus – představuje možnost různého chování objektu na stejný podnět, tedy metoda se stejným jménem může představovat jiné funkce.

Od 90. let 20. století se začali objevovat různé metody založené na objektově orientovaném přístupu, některé dnes nejpoužívanější jsou popsány níže: [12] [22]

- OMT – Object Modelling Technique, technika představující využití principů jak klasických, tak i objektových metod. Nejvíce se používá při navržení databázově orientovaných řešení;
- Coad-Yourdon Method – patří do nejstarších metod založených na objektově orientovaném přístupu. Podle autorů této metody je potřeba rozčlenit znázornění systému do několika vrstev nebo podle cíle určitých objektů. Proces vývoje systému tvoří strategie analýzy a strategie návrhu systému;
- OOSE – Object-Oriented Software Engineering je metoda založená na modelu Use Case, který spojuje všechny etapy při vývoji IS. Právě kvůli němu byla známa a tento model byl převzat dalšími metodami;
- OOER notation – metoda používající notaci ER diagramu, které byly doplněné o objektové vlastnosti. Hodí se k využívání v objektových databázích;
- CRC – Class, Responsibility, Collaborators je metoda, která má za cíl nalezení objektů v navrhovaném systému. Pro stanovení objektů u této metody se používá Use Case model spolu s dynamickým modelem. CRC pro identifikaci objektů používá nástroj daného formátu, a tím jsou tzv. indexové karty. Karty představují třídy s přiřazenými odpovědnostmi, kterým náleží objekty, jež na nich spolupracují.

3.5 CASE nástroje

CASE (Computer Aided Software Engineering) jsou sady softwarových nástrojů používaných při návrhu systému. Existují nástroje podporující jak objektově orientovaný, tak i strukturovaný přístup vývoje. CASE nástroje pokrývají celé spektrum prací na tvorbě

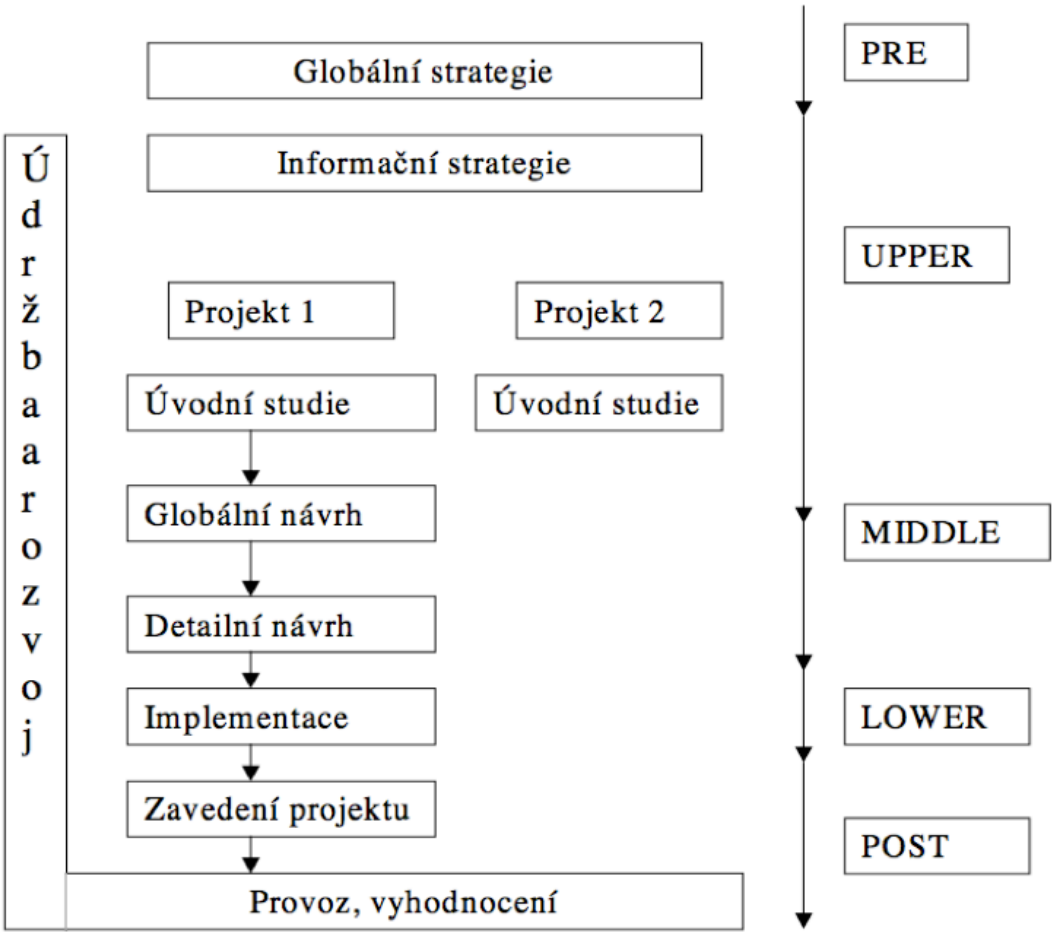
a údržbě systému, hlavně analýzu a vývoj, přípravu projektové dokumentace, kódování a testování. CASE technologie mají řadu typických vlastností, které jsou uvedené dále: [24]

- grafické nástroje pro návrh a dokumentaci modelů IS;
- organizované datové uložiště zahrnující informace o verzích a jednotlivých komponentech systému;
- rozšíření možnosti vývoje systému pomocí integrace několika komponentů CASE;
- moderní CASE nástroje podporují různé technologie modelování IS od jednoduchých metod analýzy po nástroje pro plnou automatizaci celého životního cyklu IS.

Různé CASE nástroje se používají v jednotlivých fázích a obvykle pokrývají jen určité činnosti. CASE nástroje například lze rozdělit podle životního cyklu následovně: [24]

- Pre CASE – pomáhají při tvorbě globální strategie;
- Upper CASE – soustředí se na plánování, specifikaci požadavků, modelování organizaci a globální analýzy systému. Tyto nástroje mají za cíl specifikovat systém jako celek;
- Middle CASE – podpora detailní analýzy a vlastní návrh informačního systému;
- Lower CASE – podporuje fyzickou realizaci IS;
- Post CASE – podpora organizační činnosti, tj. zavedení, údržba a rozvoj systému.

Následující obrázek ukazuje pokrytí životního cyklu vývoje softwaru CASE nástroji: [25]



Obrázek 6 – Pokrytí CASE nástroji životního cyklu vývoje (Zdroj: [25])

3.6 UML

UML neboli sjednocený modelovací jazyk je rodina notací založených na jediném metamodelu, který slouží pro popis a návrh IS, hlavně postaveném na objektově orientovaném přístupu. UML představuje relativně otevřený standard spravovaný společností OMG, která byla vytvořena za účelem vyvíjení specifikací podporujících mezisystémovou komunikaci, zejména objektově orientovaných systémů. [26]

UML pochází z metodiky objektového modelování (OMT) z roku 1991. Tato technika byla známá jako i některé další. Popularita objektově orientovaného přístupu vedla ke vzniku mnoha různých systémů notací, založených na podobných principech, ale popsaných odlišně, což vedlo ke zmatku mezi vývojáři. Bylo nutné sjednotit tyto systémy. V roce 1994 Jim Rumbaugh a Grady Booch začali pracovat nad sloučením OMT a Rational, k nimž se později přidal Ivar Jacobson s konceptem Objectory. V roce 1996 OMG (skupina správy objektů) zahájila soutěž o nejlepší notační standard pro objektově orientované modelování,

kterého se zúčastnilo několik společností. Výsledkem bylo vytvoření konečného systému za aktivní účasti Booche, Rumbaughu a Jacobsona a v roce 1997 skupina OMG anonymně přijala výsledný sjednocený modelovací jazyk jako standard. Společnosti, které se soutěže zúčastnily, převedly práva na UML na skupinu OMG. Vytvoření UML tedy vedlo k jeho přijetí jako obecně akceptovaného. [23]

Aplikovat UML v praxi se dá hodně způsoby, avšak pro jednoduchost, vyčleňují tyto tři: [26]

- UML pro náčrtky – tento způsob aplikace usnadňuje týmu komunikaci o různých částech systému. Náčrty se používají jak pro návrh komponentů, během kterého se nějakým způsobem zakreslují diagramy, třeba pomocí jednoduchých kreslících nástrojů nebo přímo na papír pro další vyjednávání s kolegy jedině o tomto komponentu, tak i pro vysvětlení fungování určitých částí zkoumaného IS;
- UML pro projektování – zde se jedná o implementační plán pro vývojáře systému, vytvořený za pomoci CASE nástrojů. Celý projekt musí být popsán co nejvíc do detailů, aby následně programátor neměl problém řešit samotný návrh, ale soustředil se na implementaci. Zde se staví s modely, ne s náčrty, které pak můžou být využity k tvoření projektové dokumentace;
- UML jako programovací jazyk – tento způsob užití UML představuje schopnost převodu UML diagramu do zdrojového kódu, který se pak dá použít pro implementaci, například modelu databáze z diagramu tříd.

UML byl vyvinut jako sjednocený standard pro záznam, konstrukci, vizualizaci a dokumentaci komponentů systémů. Definice UML zahrnuje čtyři části, které jsou stručně popsané v další tabulce: [14]

Část	Popis
Notace UML	Je potřeba mít sjednocený nastroj, který pomůže popsat statickou strukturu, funkční a dynamickou stránku systému. Právě pro to je UML, který obsahuje notaci, tedy diagramy k tomu potřebné, aby bylo zajištěno porozumění mezi všemi účastníky analýzy a návrhu IS.

Metamodel UML (sémantika)	Sémantika se zabývá popisem významu jednotlivých zápisů v UML. Její popis je členěn do čtyř vrstev, které na různé úrovni abstrakce popisují vlastnosti diagramů UML, včetně možných rozšíření. Na nejnižší úrovni jsou specifikovány vlastnosti primitivních dat. O úroveň výše je vyjádřena sémantika uživatelských objektů. Ještě výše stojí popis prvků modelu. Nejvýše je pak definice vlastností metamodelu.
Jazyk OCL	Jazyk OCL slouží pro vyjádření komplikovaných integritních omezení, popis vlastností operací a vyjádření sémantiky, které nelze popsat pomocí diagramu.
Specifikace výměnných formátů	UML má také specifikaci převodu do výměnných formátů, např. pro přenos zápisů mezi různými nástroji.

Tabulka 1 – Definice UML (Zdroj: [14])

3.6.1 Návrh IS s využitím UML

Pro návrh informačního systému pomocí UML je lepší k tomu přistoupit ze třech různých pohledů propojených navzájem mezi sebou. Každý přístup představuje jeden model, který popisuje důležité aspekty navrhovaného systému, avšak je potřeba mít všechny pro lepší zachycení podstaty navrhovaného systému. Modely nejsou úplně nezávislé, protože systém představuje víc než jen propojené části. Dobrým projektem je ten, ve kterém jsou jednoznačně jasné určité aspekty a vazby jsou nastavené do potřebné míry. Během zpracování návrhu se všechny modely rozvíjejí postupně. Ze začátku probíhá konstrukce modelu IS, nehledě na další implementaci, pak do modelu se přidávají potřebné konstrukce a dále probíhá samotná realizace. Model se ukazuje nejen svým tvarem, ale i fází vývoje. [23]

Dále budou popsány zmíněné modely, avšak diagramy uvedené v modelech budou lépe prostudovány v další části této práce.

3.6.1.1 Model tříd

Model tříd popisuje strukturu objektů IS, jejich atributy, metody a vazby s dalšími objekty. Jedná se o objektovou část. Tento model tvoří kontext pro stavový model a model

chování. Změny a interakce nedávají smysl, pokud není měnící se objekt nebo nejsou spolupracující objekty. Hlavním cílem modelu tříd je zachytit důležité koncepty potřebné pro daný systém. Model tříd se zobrazuje pomocí diagramu tříd. Třídy naznačují atributy každého objektu a metody, které provádí objekty nebo probíhají za jejich účasti. [23]

3.6.1.2 Stavový model

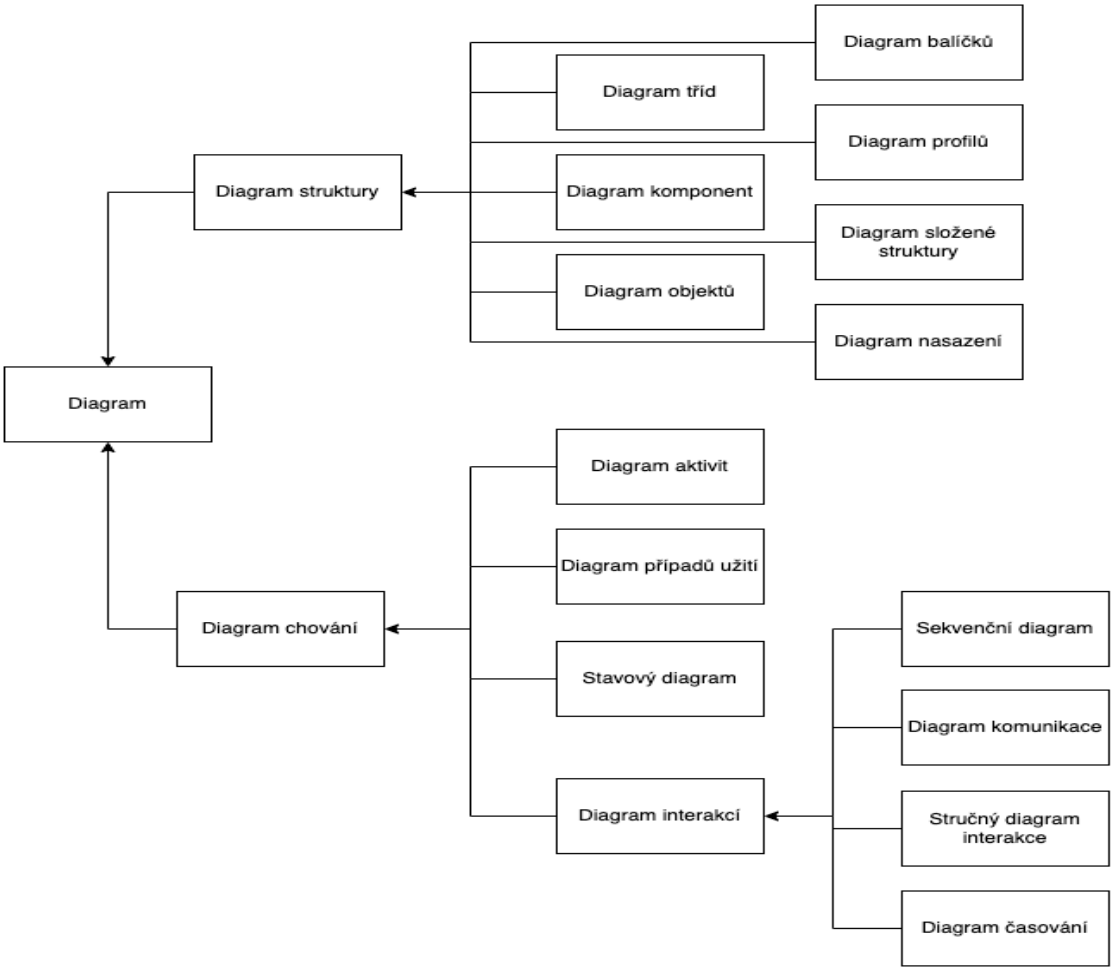
Stavový model popisuje aspekty objektů související s plynutím času a posloupností operací, tedy události spojené se změnami, stavy, které určují kontext událostí, a uspořádání událostí a stavů neboli popisuje dynamickou část systému. Stavový model pokrývá problematiku řízení – aspekt systému, který popisuje pořadí prováděných operací bez ohledu na jejich skutečný význam, účastníky a implementaci. Stavový model je zobrazen pomocí stavových diagramů. [23]

3.6.1.3 Model jednání

Model jednání popisuje interakci mezi objekty, tedy spolupráci objektů k zajištění potřebného chování systému jako celku. Stavový model a model jednání popisují různé aspekty chování a obojí je potřebné k úplnému popisu chování. Model jednání se znázorňuje pomocí případů užití, scénářů činnosti a diagramů aktivit. Případy užití popisují hlavní interakce systému s vnějšími aktéry. Scénáře činnosti ukazují časový aspekt interakce objektů spolu se samotnými objekty. Diagramy aktivit nahrazují neexistující diagramy datových toků v UML. [23]

3.6.2 Diagramy UML

Pro modelování IS pomocí UML se používají diagramy. Diagramy představují vizuální znázornění určité množiny prvků modelu systému ve formě grafu, ve kterém relace spojují entity. Ve své grafické podobě se různé typy UML diagramů používají k vizualizaci různých aspektů systému nebo jeho chování. Ve verzí UML 2.5.1 je k dispozici 14 druhů diagramů, které se dělí do dvou skupin: diagramy chování, navíc ještě obsahuje další skupinu – tj. diagram interakcí, a diagramy struktury. Klasifikace diagramů jsou zobrazena na dalším obrázku: [27] [28]

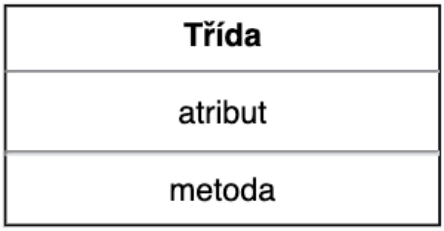


Obrázek 7 – Klasifikace diagramů UML (Zdroj: [28])

Všechny v klasifikaci zmíněné diagramy nejsou potřebné v rámci dané diplomové práce. Pro její účely postačí diagramy, které byly uvedeny u jednotlivých modelů. Následující podkapitoly budou věnovány právě těmto diagramům.

3.6.2.1 Diagram tříd

Diagram tříd slouží pro zachycení statické struktury IS. Samotný diagram představuje graf s uzly a hranami, přičemž základní stavební prvky tvoří třídy a vztahy mezi nimi. Uzly představují třídy a hrany reprezentují vztahy. Graficky třída představuje obdélník, který je členěn do tří části. Příklad je uveden na dalším obrázku: [14]



Obrázek 8 – Třída v UML (Zdroj: Vlastní zpracování)

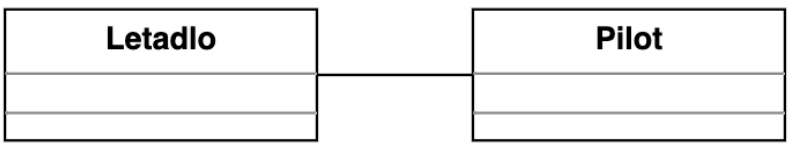
Vztahy v modelu tříd umožňují označovat vztah mezi konkrétními instancemi tříd. Také u vztahů se můžou objevovat definice role, tj. označení konce vztahu, které uvádí role objektu v daném vztahu. Vztahy mají speciální notace pro vyjádření násobností neboli kardinality, která ukazuje možný počet výskytů objektů v daném vztahu. Popis variant kardinality je k nahlédnutí v následující tabulce: [14] [29]

Notace	Popis
1	Konkrétní číslo, zde 1
*****	Libovolný počet, muže být i 0
1..*	Dvě tečky označují interval, zde od 1 do nekonečna
0..1	Označuje volitelnost, může nabývat i hodnoty NULL
Žádné označení	Pokud násobnost není uvedena, označuje to výchozí hodnotu 1

Tabulka 2 – Notace a popis kardinality v diagramu tříd (Zdroj: Vlastní zpracování)

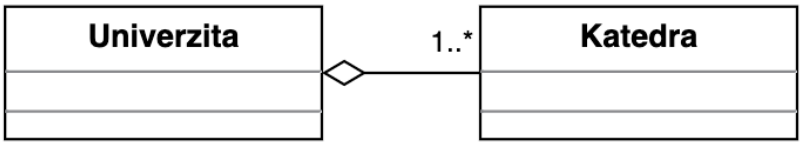
Model tříd disponuje čtyřmi typy vazeb mezi třídami: [23] [26] [27] [29]

- Asociace – představuje základní spojení, tedy fyzickou nebo konceptuální souvislost mezi instancemi objektů, které můžou existovat nezávisle na sobě. Obvykle je tato vazba dvousměrná, což znamená, že obě třídy mají odkaz na druhou. Asociace se zakresluje jako plná čára. Příklad asociace je uveden na dalším obrázku:



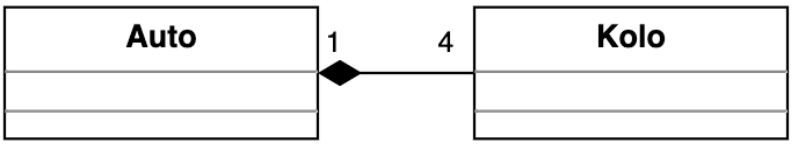
Obrázek 9 – Ukázka asociace v diagramu tříd (Zdroj: Vlastní zpracování)

- Agregace – představuje vztah typu část-celek, který se značí plnou čárou a končí na straně celku prázdným kosočtvercem. Celek představuje entitu, která drží kolekci prvku. Entita představující část, muže existovat i sama o sobě, tedy agregace může představovat vztah dvou typů: „*skládá se*“ nebo „*je součástí*“. Agregace by mohla vypadat následovně:



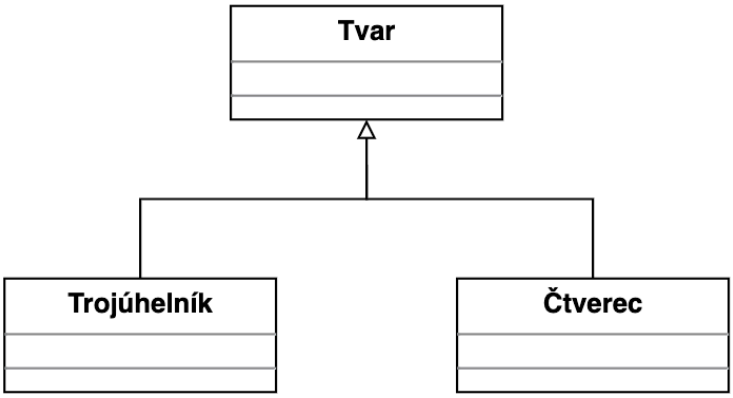
Obrázek 10 – Ukázka agregace v diagramu tříd (Zdroj: Vlastní zpracování)

- Kompozice – představuje speciální případ agregace, avšak je silnější a platí dvě omezení: část patří pouze jednomu celku a když zaniká celek, zaniká i část. Graficky se zobrazuje jako plná čára, která končí plným kosočtvercem. Příklad kompozice je uveden na následujícím obrázku:



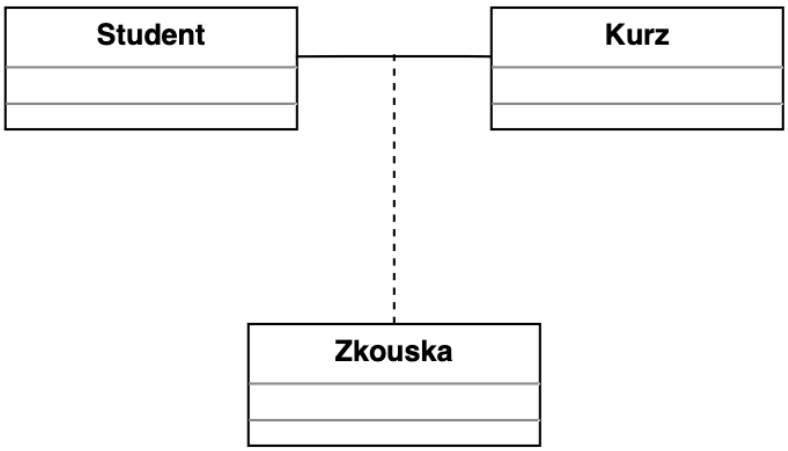
Obrázek 11 – Ukázka kompozice v diagramu tříd (Zdroj: Vlastní zpracování)

- Generalizace – představuje vztah pro sdílení podobností mezi třídami při zachování jejich odlišností, např. se jedná o dědičnost. Jedna třída, která se nazývá podtřída nebo následník, dědí vlastnosti jiné, které se říká nadtřída nebo předek. Obvykle podtřída přidává své vlastní atributy a metody spolu s těmi, které dědí od svých předků, tedy dědičnost je přenosná přes libovolný počet úrovní. Zakresluje se jako plná čára, která končí na jedné straně třídy, ze které se dědí, jež se zakresluje prázdným trojúhelníkem. Další obrázek ukazuje generalizaci mezi třídami:



Obrázek 12 – Ukázka generalizace v diagramu tříd (Zdroj: Vlastní zpracování)

V UML také existuje speciální typ třídy, který se jmenuje asociační třída. Ta představuje vazbu mezi dvěma dalšími entitami. Používá se, když je potřeba přidružit další vlastnosti nebo chování k asociaci, spíše než k třídám na obou koncích vazby. Asociační třída je v podstatě prostředníkem mezi dvěma třídami s atributy a metodami specifickými pro asociaci. Označuje se jako třída, která je propojena přerušovanou čárou jdoucí od asociace mezi třídami, viz další obrázek s příkladem: [23]

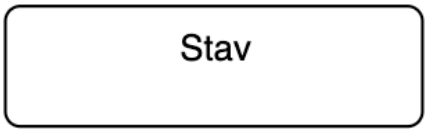


Obrázek 13 – Příklad asociační třídy v UML (Zdroj: Vlastní zpracování)

3.6.2.2 Stavové diagramy

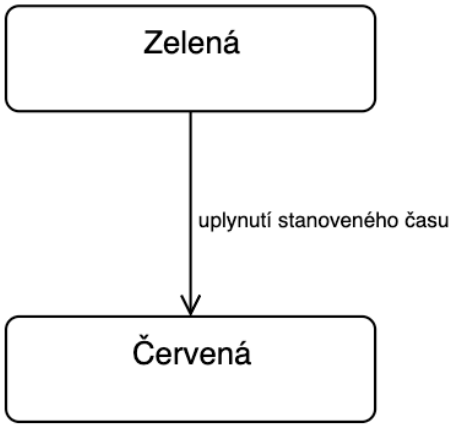
Stavové diagramy jsou typem diagramů chování, které se používají k modelování chování systému nebo jeho částí v průběhu času, tedy popisuje dynamiku systému. Používají se k reprezentaci různých stavů, ve kterých se systém nebo jeho část může nacházet, událostí nebo akcí způsobujících přechod mezi stavy a podmínek nebo omezení, která musí být splněna, aby k přechodu došlo. Stavový diagram v UML se skládá ze tří hlavních prvků: stavů, přechodů a událostí. [23] [27] [29]

Stav je graficky znázorněn jako ovál. Představuje různé podmínky nebo režimy, ve kterých se může systém nebo část nacházet. Stav lze dále rozdělit na podstavy, které jsou reprezentovány menšími obdélníky v rámci hlavního stavového obdélníku. V UML stav vypadá následovně: [27] [29]



Obrázek 14 – Grafické zobrazeni stavu v UML (Zdroj: Vlastní zpracování)

Přechody jsou znázorněny šipkami a jsou označeny událostí, nebo akcí, která způsobí, že k přechodu dojde. Spojují dva stavy a indikují, jak se systém nebo jeho část přesouvá z jednoho stavu do druhého, tedy přechod je změna stavu způsobená událostí. Příklad přechodu mezi dvěma stavy je zobrazen na dalším obrázku, který ukazuje změnu barvy světla po uplynutí určitého času: [27] [29]



Obrázek 15 – Příklad přechodu mezi stavy (Zdroj: Vlastní zpracování)

Kromě stavů, přechodů a událostí mohou stavové diagramy zahrnovat také aktivity, které představují odezvu na stavy a události objektů. Aktivita může probíhat na začátku, konci, uvnitř stavu nebo na přechodu. Aktivity lze rozdělit do dvou typu: [29]

- do-aktivita – představuje operaci obsahující časové trvání, a která je přiřazena ke stavu a k přechodu přiřazena být nemůže. Do-aktivity zahrnují operace, které trvají bez omezení a operace, které po uplynutí určitého času samovolně skončí. Notace je: „do/název aktivity" v symbolu stavu;

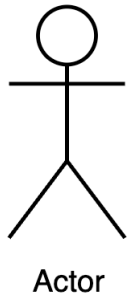
- vstupní a výstupní aktivity – představují aktivity, které probíhají na začátku nebo konci stavu a uvádějí se uvnitř. Vstupní aktivity mají notaci: „entry/název aktivity“, výstupní: „exit/název aktivity“.

Také stavové diagramy obsahují notace pro označení počátečního a konečného bodu, který ukazuje začátek a konec daného stavu.

3.6.2.3 Diagram případů užití

Diagram případu užití, taktéž známý jako Use Case Diagram, se používá k reprezentaci dynamického chování systému. Odráží funkčnost systému pomocí případů užití, aktérů a jejich vztahů. Modeluje úlohy, služby a funkce požadované systémem nebo jeho části. Zobrazuje funkčnost systému na vysoké úrovni a také říká, jak by měl uživatel se systémem zacházet. Hlavním účelem užití diagramového případu je zobrazit dynamickou stránku systému. Shromažďuje systémové požadavky, které zahrnují vnitřní i vnější vlivy. Vyvolává aktéry, případy užití a některé věci, které volají aktéry a prvky odpovědné za implementaci diagramů případů užití. Představuje, jak může entita z vnějšího prostředí interagovat s částí systému. Základní prvky diagramu případu užití tvoří: [14] [30]

- aktér – jde o uživatelskou roli nebo spolupracující systém, jednoduše lze říct, že se jedná o abstraktního uživatele daného systému. Graficky aktér vypadá následovně:



Obrázek 16 – Aktér v notaci UML (Zdroj: Vlastní zpracování)

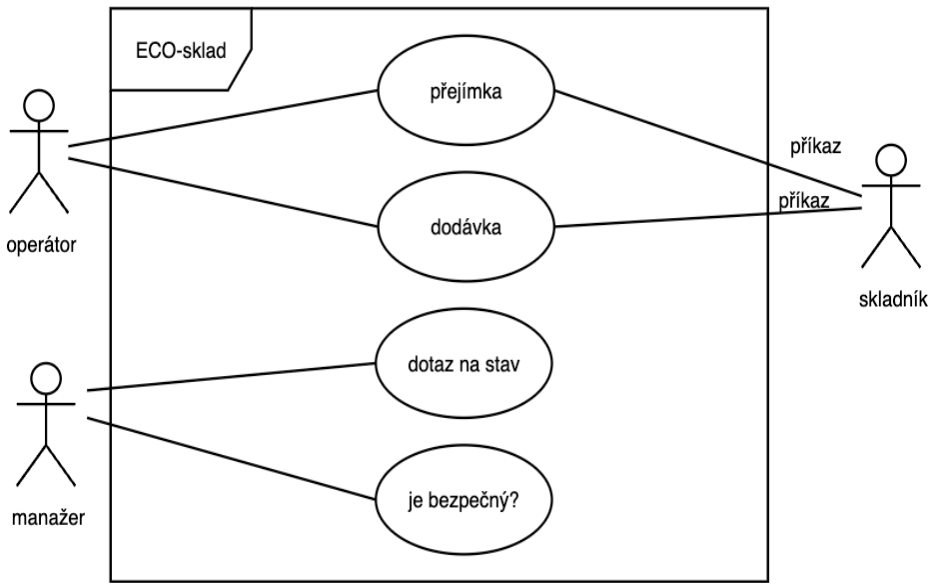
- hranice systému – jde o označení hranice mezi systémem a jeho okolím. Graficky představuje obdélník označený navěštím a zahrnuje případy užití. Vše, co je uvnitř, představuje oblast spadající do kompetence systému. Elementy mimo označenou oblast představují okolí;

- případ užití („use case“) – je souvislou částí funkčnosti, kterou systém poskytuje aktérům prostřednictvím interakce s nimi. Graficky se znázorňuje jako elipsa, která vevnitř má popis odpovídající aktivitě:

Obrázek 17 – Případ užití v UML (Zdroj: Vlastní zpracování)

- komunikace – je vazba mezi aktérem a use case, tedy komunikace aktéra a daným IS na konkrétním případu užití.

Na dalším obrázku je uveden diagram případu užití, který popisuje hranice systému označeného jako ECO-sklad, také jsou zde dva typy aktérů. V rámci diagramu případu užití se používají sekundární aktéři, např. skladník, který představuje uživatelskou roli nebo spolupracující systém nutný pro činnost daného IS. [14]



Obrázek 18 – Ukázka digramu případu užití (Zdroj: [14])

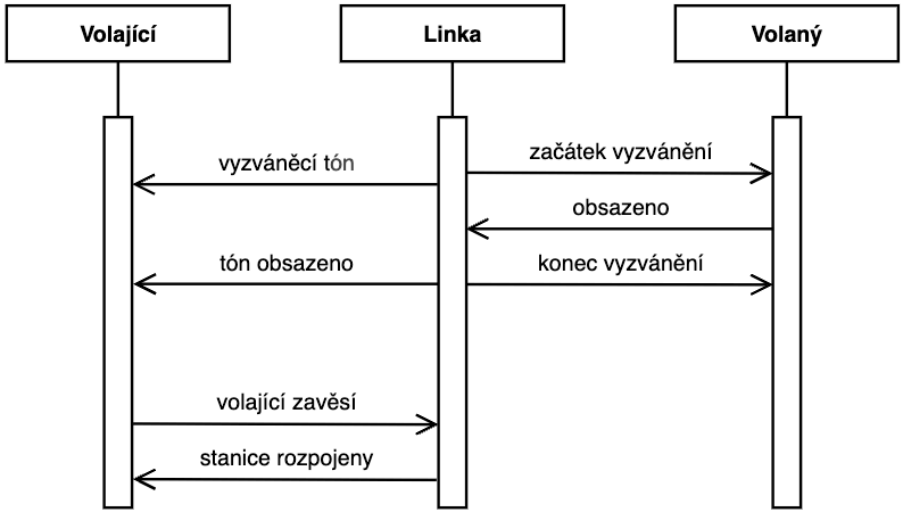
Pro explicitní vyznačení směru vazby se dá použít orientovanou šipku. Pro explicitní vyjádření vztahu mezi případy užití, spolu s orientovanou přerušující šipkou, se používají stereotypy. UML má dva předdefinované: [14]

- <<include>> – ukazuje, že jeden use case zahrnuje druhy;
- <<extend>> – ukazuje, že nějaký případ rozšiřuje chování.

3.6.2.4 Scénáře činností

Scénáře činností, které jsou také známé jako sekvenční diagramy. Pro určitou množinu elementů na jedné časové ose zobrazují životní cyklus objektu (od tvorby přes jeho činnost a následné zaniknutí) a interakci aktérů IS v rámci případu užití. Scénář činností tvoří dva základní prvky: [31] [29]

- scénář – představuje posloupnost činností, které budou probíhat v rámci určitého use case. Obvykle mají textovou reprezentaci, avšak mají různý rozsah. Scénář je tvořen zprávy mezi objekty a aktivity, které provádí;
- sekvenční diagram – zobrazuje účastníky interakce a zprávy mezi nimi. Aktér je zobrazen pomocí sloupce, kterému se říká „pupeční šňůra“ a zprávy jsou představeny jako horizontální šipky mezi odesilatelem a příjemcem dané zprávy. Objekty si můžou volat i samy sobě, jedná se o tzv. self-call. Čas jde směrem shora dolu, takhle se odráží pořádek zpráv. Pro každý případ užití má být vytvořen jeden, nebo více sekvenčních diagramů. Sekvenční diagram může obsahovat i tzv. pasivní objekty, které se aktivují až jsou zavolány, až jejich události proběhnou, tak se znovu stávají neaktivními a zpátky předávají řízení volajícímu objektu. Místo sloupce má neaktivní element jako symbol přerušovanou čáru a v době, kdy se stává aktivním, tak se změní na obdélníkový pruh. Taktéž existují dočasné objekty, které existují od zavolání do ukončení své aktivity, kde X značí zaniknutí. Příklad jednoduchého sekvenčního diagramu je zobrazen na následujícím obrázku:



Obrázek 19 – Příklad sekvenčního diagramu v UML (Zdroj: [29])

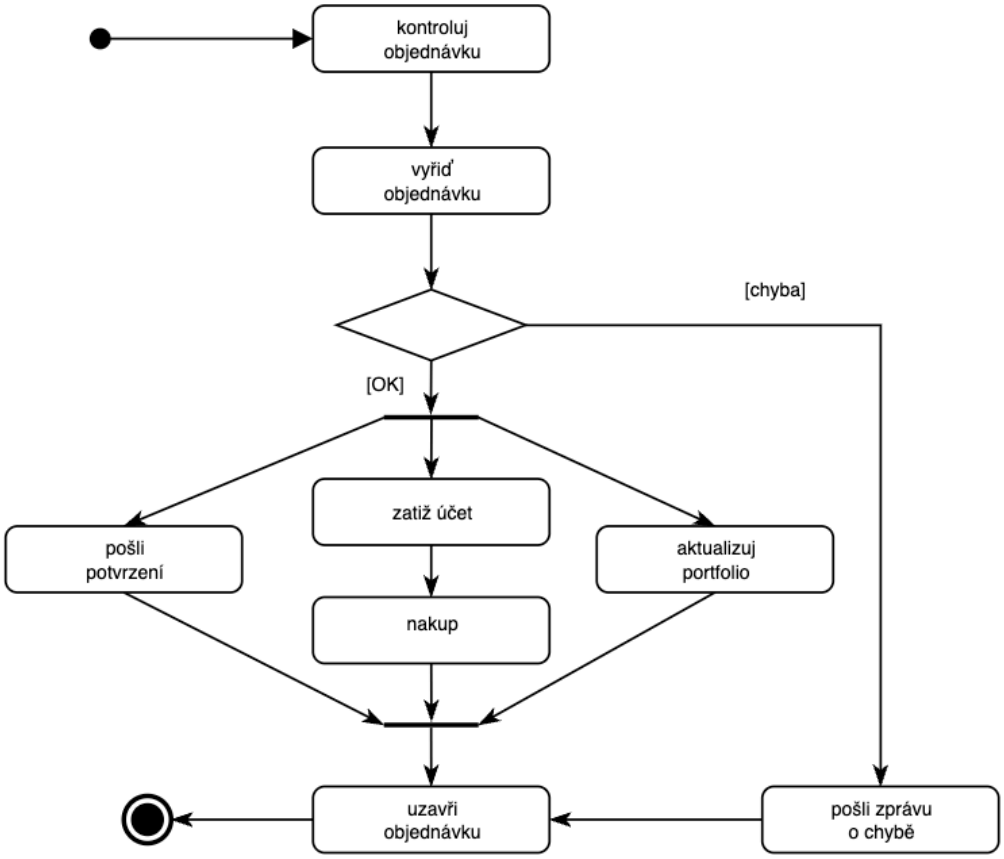
3.6.2.5 Diagramy aktivit

Diagramy aktivit se používají k modelovaní toků aktivit v rámci systému, tím nahrazují neexistující diagramy datových toků v UML. Napodobuje stavovému diagramu, ale navíc umožňuje modelovat tzv. aktivity. Aktivita zde představuje synchronní operaci, tedy přechod k další aktivitě se začíná až proběhne aktivita předchozí. Diagramy aktivit se používají pro dokumentaci tzv. workflow. UML notace obsahuje následující prvky: [14] [29]

- aktivita – protáhlý ovál s vepsaným názvem;
- rozhodovací bod – „diamant“, který představuje bod rozhodnutí, kde je další tok vázán na přidruženou podmínku;
- pokračování – šipka;
- štěpení a spojování větví – tlustá příčka;
- začátek – plné „kolečko“;
- konec – „volské oko“.

Vetší část aktivit se až dokončí svoji činnost zastaví, ale některé aktivity nejsou časově omezené, dokud nebudou pozastaveny vnější událostí. Většinou dokončení aktivity znamená, že další aktivita se může začít. Šipka, která je nepojmenovaná, směřující od jedné aktivity k jiné, označuje, že první má skončit před začátkem druhé. Po štěpení, kdy se aktivitám říká konkurenční, můžou probíhat v libovolném pořadí, avšak před sdružením musí být dokončeny všechny. Pokud aktivita pokračuje několika dalšími aktivitami, musí

být každá větev popsána podmínkou v hranatých závorkách. Příklad jednoduchého diagramu aktivity je zobrazen na dalším obrázku: [29]



Obrázek 20 – Příklad diagramu aktivit (Zdroj: [29])

4 Vlastní práce

Následující část dané diplomové práce bude věnovaná vlastní analýze a návrhu informačního systému. Aktuální situace si žádá vylepšení a rozšíření funkcionality systému stávajícího, který představuje interní bankovní systém pro prohlížení dokumentů a následnou správu v externím IS. Po pohovorech s vedením, koncovými uživateli a vedoucím programátorem bylo rozhodnuto zavést nový systém na bázi systému současného.

Jako první krok bude provedena analýza současného stavu IS, spolu s definováním kritických závislostí. Následně budou formulovány požadavky na informační systém, které budou získány pomocí pohovorů s uživateli, vedením a vedoucím programátorem.

Dále bude navržen nový informační systém pomocí jazyka UML, a to z hlediska tří modelů: modelu tříd, stavového modelu a modelu jednání. Model tříd bude představen diagramem tříd, který bude reprezentovat statickou strukturu systému. Stavový model bude tvořen stavovými diagramy, které slouží pro popis chování systému. Model jednání bude složen z případů užití, scénářů činností a diagramů aktivit. Případy užití popisují hlavní interakce systému s vnějšími aktéry. Scénáře činností ukazují časový aspekt interakce objektů spolu se samotnými objekty. Diagramy aktivit také jako stavové diagramy popisují chování systému. K návrhu bude použít CASE nastroj StarUML, se kterým má autor práce předchozí zkušenosti, což je důvod, proč byl vybrán.

4.1 Analýza současného stavu IS

SMART je interní systém pro prohlížení dokumentů, který je založen na knihovně od společnosti SAP. Všechny dokumenty vznikají na bázi databázového řešení od SAP, které dále nabízí modul pro správu a tvorbu složek s dokumenty.

Aktuální verze systému byla naposled představena v roce 2012. Podněty pro zavedení nového systému byly: kritická zranitelnost log4j (logovací modul pro jazyk Java), která se objevila v roce 2021 a umožňuje vzdálené spuštění zdrojového kódu. Dále kritická závislost knihovny Apache Struts, jež ulehčuje práci s architekturou MVC, na které je postaven současný systém, možnost propojení modulu správy dokumentů s daným IS v novějších verzích knihovny SAP a také zastaralý design nesplňující bankovní standard.

V současnosti systém představuje katalog složek, které následně lze otevřít pro prohlédnutí dokumentů, jež tato složka zahrnuje. Také dokumenty můžou obsahovat záznam instance, představující informaci a verzi dokumentu před poslední změnou, tedy uživatel může prohlédnout provedené změny spolu s časem a osobou, která to provedla, a má k této verzi i přístup. Zatím systém neumožňuje správu v rámci jednoho okna, pokud uživatel potřebuje provést změnu, tak se musí přihlásit k dalšímu systému a otevřít potřebný dokument nebo instanci pomocí unikátního čísla, avšak systém podporuje stahování.

Ze strany softwaru systém obsahuje kritické zranitelnosti, a to log4j a Apache Struts, knihovna, kterou nejde, jako v případě log4j, jednoduše obnovit, je potřeba komplexního předělání větší části zdrojového kódu stávajícího systému, což je dalším důvodem k zavedení systému nového, za použití jiné knihovny jmenovitě Spring Framework.

4.2 Specifikace požadavků

Specifikace požadavků na informační systém je nezbytná pro úspěšný proces vývoje a pro zajištění toho, aby výsledný systém vyhovoval potřebám jeho uživatelů a zainteresovaných stran. Specifikace požadavků na IS je důležitá z několika důvodu:

- přehlednost – požadavky pomáhají jasně definovat účel a cíle informačního systému. To pomáhá zajistit, aby každý člověk zapojený do procesu vývoje jasně chápal, co by měl systém dělat a co se od něj očekává;
- vylepšený vývoj – specifikace požadavků umožňuje vývojářům vytvořit přesný návrh a plán informačního systému, což snižuje pravděpodobnost nedorozumění nebo změn během procesu vývoje;
- lepší uživatelský dojem – požadavky pomáhají zajistit, aby systém vyhovoval potřebám uživatelů, kteří jej budou používat. To vede k lepší uživatelské zkušenosti, protože systém je od začátku navržen s ohledem na uživatele;
- zvýšená efektivita – požadavky usnadňují identifikaci případných mezer ve schopnostech informačního systému;
- zlepšená kvalita – požadavky poskytují základ pro testování a hodnocení informačního systému. Zadáním toho, co by měl systém dělat, je možné jej otestovat, aby bylo zajištěno, že splňuje požadované standardy;

- lepší řízení projektu – požadavky slouží jako plán pro proces vývoje. Definováním cílů informačního systému a kroků potřebných k jejich dosažení mohou projektoví manažeři efektivněji plánovat, alokovat zdroje a sledovat pokrok.

Pro zavedení efektivního informačního systému je důležité specifikovat požadavky dvou druhů, funkcionálního a nefunkcionálního. Funkční požadavky popisují konkrétní úkoly, které musí informační systém plnit. Definují, co by měl systém dělat a jsou vyjádřeny konkrétními, měřitelnými a ověřitelnými popisy. Nefunkční požadavky popisují kvalitativní atributy, které musí mít informační systém. Tyto požadavky nejsou konkrétními úkoly, ale jsou stále důležité pro úspěšné fungování informačního systému.

Tato část právě bude obsahovat funkční a nefunkční požadavky kladené na nový informační systém. Specifikace probíhala pomocí pohovoru s vedením, uživateli stávajícího systému a vedoucím programátorem. Pro lepší přehled bude specifikace představena. Funkční požadavky, včetně popisu, jsou k dispozici v následující tabulce:

Číslo	Požadavek	Popis
1	Uživatel se přihlásí automaticky	Uživatel se přihlašuje do systému bez zadání uživatelského jména a hesla. Pomocí speciálního síťového protokolu odešle své údaje při otevření IS, které budou následně ověřeny. Avšak starý způsob přihlášení by měl být zachován.
2	Uživatel vidí záznam složek	Přihlášený uživatel má navigační panel, který obsahuje složky pro něho dostupné.
3	Uživatel spravuje dokumenty	Uživatel po kliknutí na jednotlivou složku uvidí záznam dokumentů, které daná složka zahrnuje. Po kliknutí na dokument se otevře nové okno s modulem externího systému pro správu dokumentu, bez nutnosti zadávat přihlašovací údaje uživatele a identifikátoru dokumentů. Také určité typy dokumentů by měly být spravovány administrátory systému na základě požadavků.
4	Uživatel stahuje dokumenty	Složky můžou obsahovat dokumenty různých formátů, které nelze spravovat, ale jenom stahovat.
5	Uživatel prohlíží instance	Pokud dokument byl v minulosti změněn, rozsvítí se tlačítko s historií, které po kliknutí vede na záznam změn daného dokumentu. Při kliknutí na

		jednotlivou instanci se v novém okně otevře modul spravování bez moznosti změn k nahlédnutí uživatele.
6	Administrátor spravuje obsah složek	Administrátor odpovídá za obsah složek. Při požadavku od uživatele na dodání nebo nahrání nových dokumentů odpovídá za realizaci.
7	Tvoří se logovací reporty	Otevřením jakéhokoliv dokumentu tvoří uživatel logovací report. Do databáze systému budou uloženy údaje o osobě, čase, identifikátoru a názvu otevřeného dokumentu.
8	Administrátor nahrává dokumenty	Administrátor nahrává dokumenty, které následně bude spravovat nebo jenom stahovat uživatel.

Tabulka 3 – Funkční požadavky na IS (Zdroj: Vlastní zpracování)

Další tabulka bude obsahovat nefunkční požadavky a jejich stručný popis:

Číslo	Požadavek	Popis
1	Systém musí mít nejnovější verzi logovacího modulu log4j	Systém by měl být zbaven kritické zranitelnosti spojené s logovacím modulem, a to tak, že bude obnoven do nejnovější dostupné verze.
2	Systém musí být postaven na knihovně Spring	Současný systém běží za pomoci nepodporované knihovny, která musí být nahrazena jinou, podporovanou ze strany banky.
3	Systém musí fungovat v prohlížeči Edge	Systém by měl fungovat bez problému v prohlížeče od společnosti Microsoft, který je oficiálně podporován bankou.
4	Systém má být schopen dále se rozšiřovat a snadno modifikovat	Systém má být připraven i k dalším možným výskytům kritické zranitelnosti nebo vyřazení nějakého softwarového komponentu z podpory, a proto má být schopen rychle se přizpůsobit potřebným změnám.
5	Systém má podporovat zabezpečenou komunikaci	Systém musí podporovat zašifrovanou komunikaci, aby byl schopen přihlašovat uživatele bez zadání jména a hesla do políček.
6	Systém má být responzivní pro laptopy a mít design podle bankovního standardu	V současné době stále velké množství lidí pracuje z domova, kde nemají velkou obrazovku jako v kanceláři, proto je potřeba mít responzívní design pro tyto uživatele. Také samotný design má být v souladu se standardy banky, tzn. podle daných barev, stylu písma atd.

Tabulka 4 – Nefunkční požadavky na IS (Zdroj: Vlastní zpracování)

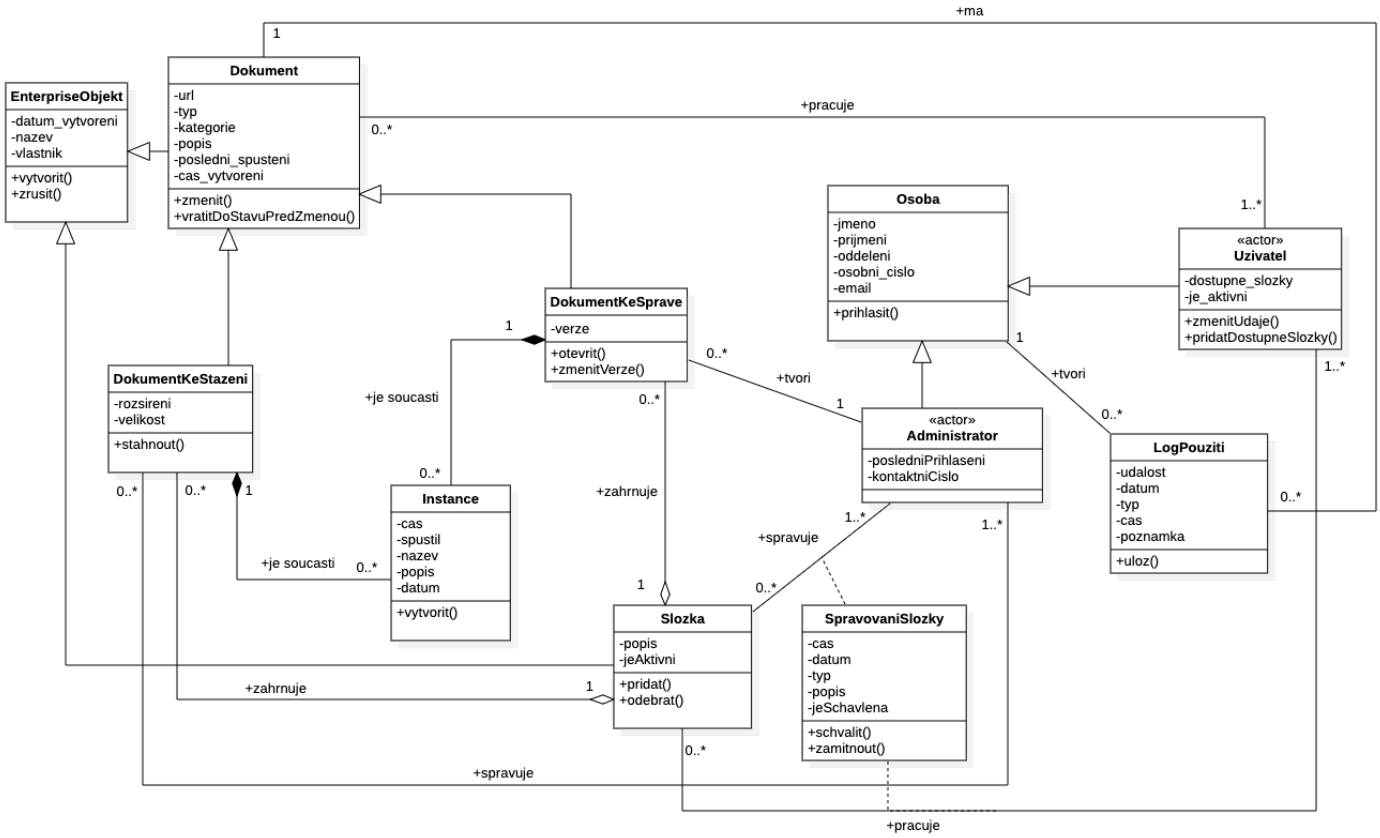
4.3 Návrh informačního systému

Tato kapitola bude zaměřena na samotný návrh IS. Návrh informačního systému je kritickou fází procesu vývoje. Jedná se o transformaci požadavků a cílů systému do podrobného plánu budování systému. UML je široce používaný vizuální jazyk pro modelování a navrhování informačních systémů, poskytuje standardizovaný způsob reprezentace složek systému a vztahů mezi nimi. Pravě UML bude použít pro návrh IS v rámci této diplomové práce. Použitím UML při návrhu lze vytvořit efektivní a účinný IS, který je dobře strukturovaný a snadno pochopitelný. Pro návrh budou použity tyto modely: model tříd, stavový model a model jednání.

4.3.1 Model tříd

Model tříd je v návrhu základem informačního systému vytvořeného pomocí jazyka UML, vytváří výchozí bod pro následnou implementaci daného IS. Model tříd udává statickou stránku systému a je představen pomocí diagramu tříd, který ukazuje jednotlivé třídy a vazby mezi nimi. Také diagram tříd bude doplněn datovým slovníkem, popisujícím třídy, jejich vlastnosti a vazby s dalšími elementy systému. Samotný diagram bude obsahovat jen důležité aspekty navrhovaného systému, jelikož zachycení všech možných prvků je nad rámec dané diplomové práce.

4.3.1.1 Diagram tříd



Obrázek 21 – Diagram tříd (Zdroj: Vlastní zpracování)

4.3.1.2 Datový slovník

EnterpriseObjekt je abstraktní třídou, představující základní prvek knihovny SAP a v rámci navrhovaného IS je předchůdcem pro třídy **Dokument** a **Složka**. Obsahuje společné atributy a základní metody jako vytvořit a zrušit pro potomky komponenty.

Dokument je abstraktní třída, potomek třídy **EnterpriseObjekt**. Jedná se o komponent systému, se kterým pracují uživatele a má dvě realizace. Obsahuje sdílené údaje pro potomky a má objekty typu **LogPouziti**, které vznikají při různých operacích s objekty tříd realizujících danou abstraktní třídu, tedy **DokumentKeStazeni** a **DokumentKeSprave**.

DokumentKeStazeni představuje potomka třídy **Dokument** a jedná se o dokumenty, které uživatelé můžou jenom stahovat pro následnou práci. Navíc ke sdíleným atributům a metodám obsahuje vlastní atributy: rozšíření ukazující rozšíření daného dokumentu, např.

.pdf nebo .xls, a také velikost tohoto souboru. Spravování těchto dokumentu by mělo probíhat na požadavek, který pak má na starosti **Administrator,** musí rozhodnout, jestli ho schválí, pokud ano, odpovídá za realizaci, také může měnit obsah i sám.

DokumentKeSprave je potomkem abstraktní třídy **Dokument**. Představuje dokumenty, které uživatel může prohlédnout a spravovat přes externí modul, což je část jiného systému.

Instance je třídou, která je součástí dokumentu a představuje jednotlivé změny, které byly provedeny u daného dokumentu, buď z důvodu změny přes Správu Dokumentů provedenou uživatelem, nebo při změně dokumentu ke stažení provedenou administrátorem. Obsahuje údaje o změně a osobě odpovědné za ně. Instance existuje pouze spolu s dokumentem, ke kterému patří, když zaniká dokument, zanikají i všechny instance.

Složka je podtřídou třídy **EnterpriseObjekt**. Jedná se o uložiště obsahující dokumenty obou druhů. Také složka může být prázdná, pokud byla vytvořena, ale ještě nemá žádné přiřazené dokumenty. Drží atributy o obsahu a metody přidat a odebrat, sloužící pro zarážení či vyrážení dokumentů.

Osoba je abstraktní třídou, která je předchůdcem pro uživatele a administrátory daného systémů. Zahrnuje atributy spojené s osobními informacemi a také metodu přihlásit, která by měla sloužit pro přihlášení do systému.

Uživatel je potomkem třídy **Osoba** a představuje řádného uživatele IS. Má pole dostupných složek, se kterými následně pracuje, pokud je jeho účet aktivní. Pro práci se systémem musí být přihlášen a mít ověřena práva přístupu k danému IS.

Administrátor je podtřídou třídy **Osoba**. Jedná se o uživatele s větším počtem funkcí a odpovědností. Administrátor může spravovat uživatelské účty, tedy přiřazovat složky a odpovídá za zpracování uživatelských požadavků. Navíc se stará o obsah dokumentů ke stažení a jejich potřebnou změnu podle požadavků řádných uživatelů. Také zakládá nové dokumenty ke správě. Pro práci se systémem musí být přihlášen a mít ověřena práva.

LogPoužiti je třída odpovědná za logování při práci s dokumenty. **LogPouziti** tvoří osoby během práce s dokumenty, tedy pokud některý dokument byl otevřen ke správě, stažen, změněn **Administratorem** atd. vytvoří se záznam, který je zatím uložen do databáze IS. Má atributy o dané události a metodu pro uložení.

SpravovaniSlozky je asociační třída zprostředkující vztah mezi entitami **Slozka**, **Administrator** a **Uzivatel**, uchovává atributy o počtu dokumentů před provedenou změnou

a časem, kdy tato událost nastala spolu s popisem a typem dané změny, pokud se jedná o požadavek od **Uzivatele**, a jestli byla schválena, či nikoliv.

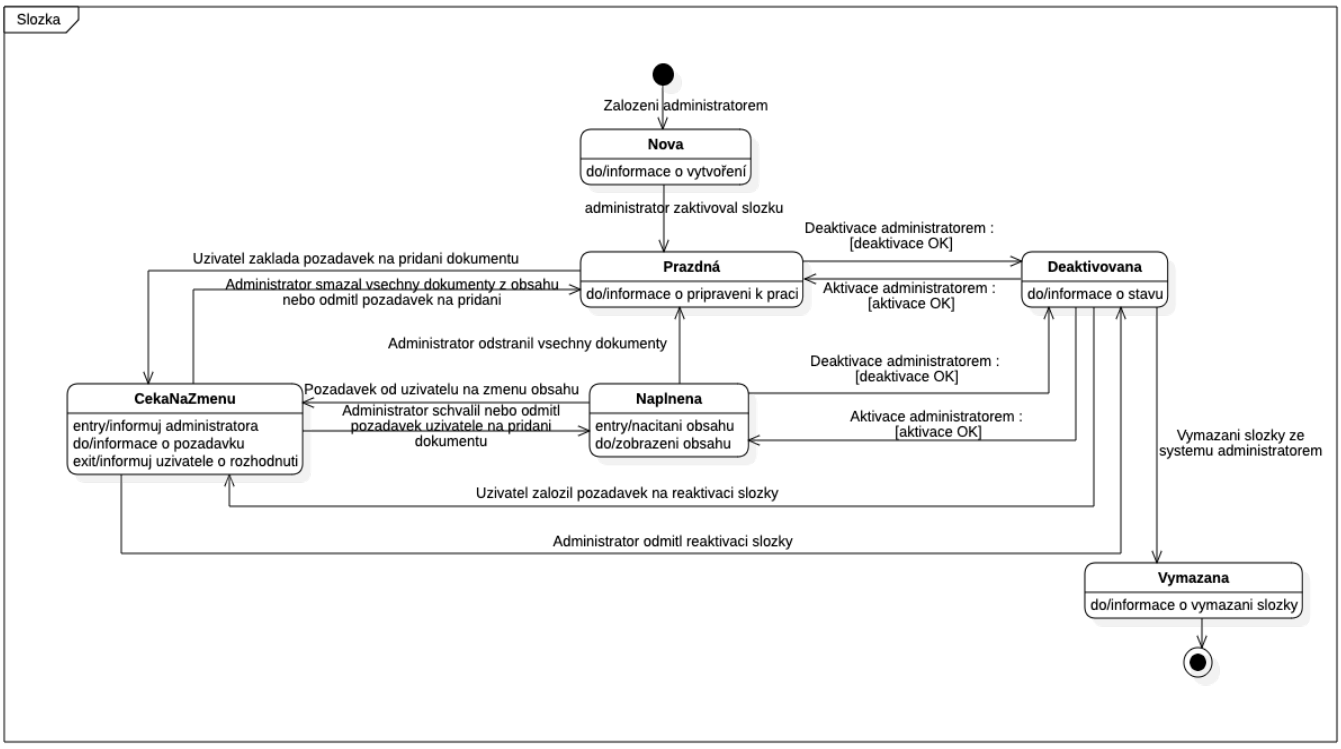
4.3.2 Stavový model

Stavový model v UML slouží pro představení chování objektu nebo systému v průběhu času. Ukazuje různé stavy, ve kterých se objekt nebo systém může nacházet a události, které spouštějí přechody mezi těmito stavy. Stavový model je znázorněn pomocí stavového diagramu, který se skládá ze zaoblených obdélníků (představují stavy), šipek (představují přechody) a štítků (popisují události a akce spojené s přechody).

Další kapitoly budou obsahovat stavové diagramy vybraných tříd potřebných pro dostatečný přehled chování systému.

4.3.2.1 Stavový diagram Složka

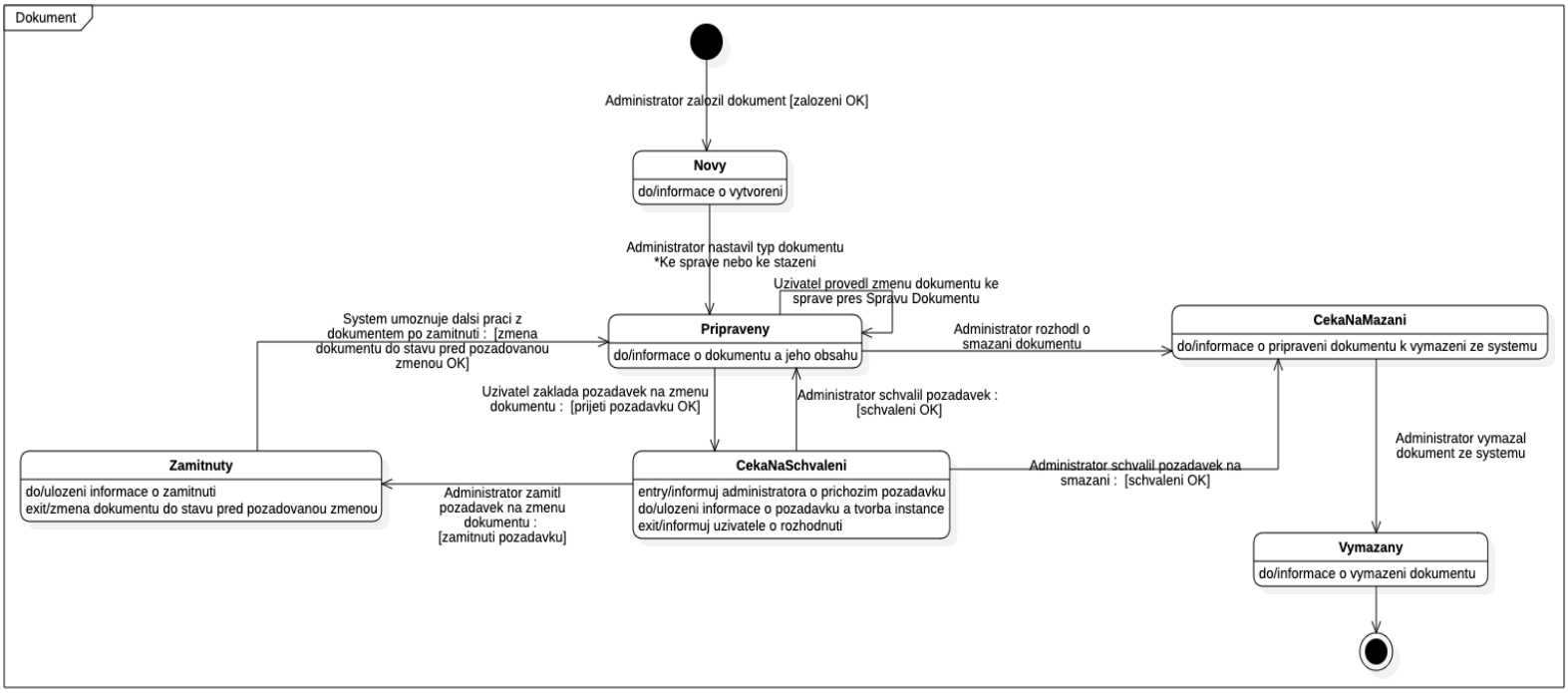
Složka vzniká založením administrátorem. Následně ji administrátor musí zaktivovat. Po první aktivaci je složka prázdná. Zatím ji administrátor spravuje, tedy upravuje obsah, a přiděluje uživatelům, podle jejich práv. Na požadavek od uživatele může složka měnit svůj obsah. Také složka může být ze systému vymazána nebo deaktivována, při tom dokumenty se zůstávají. Stavový diagram pro třídu Složka je uveden na dalším obrázku:



Obrázek 22 – Stavový diagram třídy Složka (Zdroj: Vlastní zpracování)

4.3.2.2 Stavový diagram Dokument

Dokument, stejně jako složka, vzniká založením administrátorem. Až je vytvořen, dokumentu se nastaví typ (ke stažení nebo ke správě) a tím přichází do stavu „připravený“ a je připraven k práci. Zatím, pokud dokument byl změněn uživatelem přes Správu Dokumentů, znovu přichází do stavu „připravený“. Když přišel požadavek na změnu dokumentu ke stažení, přechází dokument ze začátku do stavu „čeká na schválení“, kde administrátor musí rozhodnout o schválení požadované změny. Ve chvíli, kdy je změna schválena, přichází dokument do stavu „připravený“, jinak při odmítnutí do stavu „zamítnutý“, ze kterého se vrací do „připravený“. Pokud již není potřeba mít dokument v systému, jde do stavu „čeká na mazání“, ve kterém je označen ke smazání a potom do stavu „vymazaný“, až je vymazán administrátorem. Stavový diagram pro třídu Dokument je k nahlížení na následujícím obrázku:



Obrázek 23 – Stavový diagram třídy Dokument (Zdroj: Vlastní zpracování)

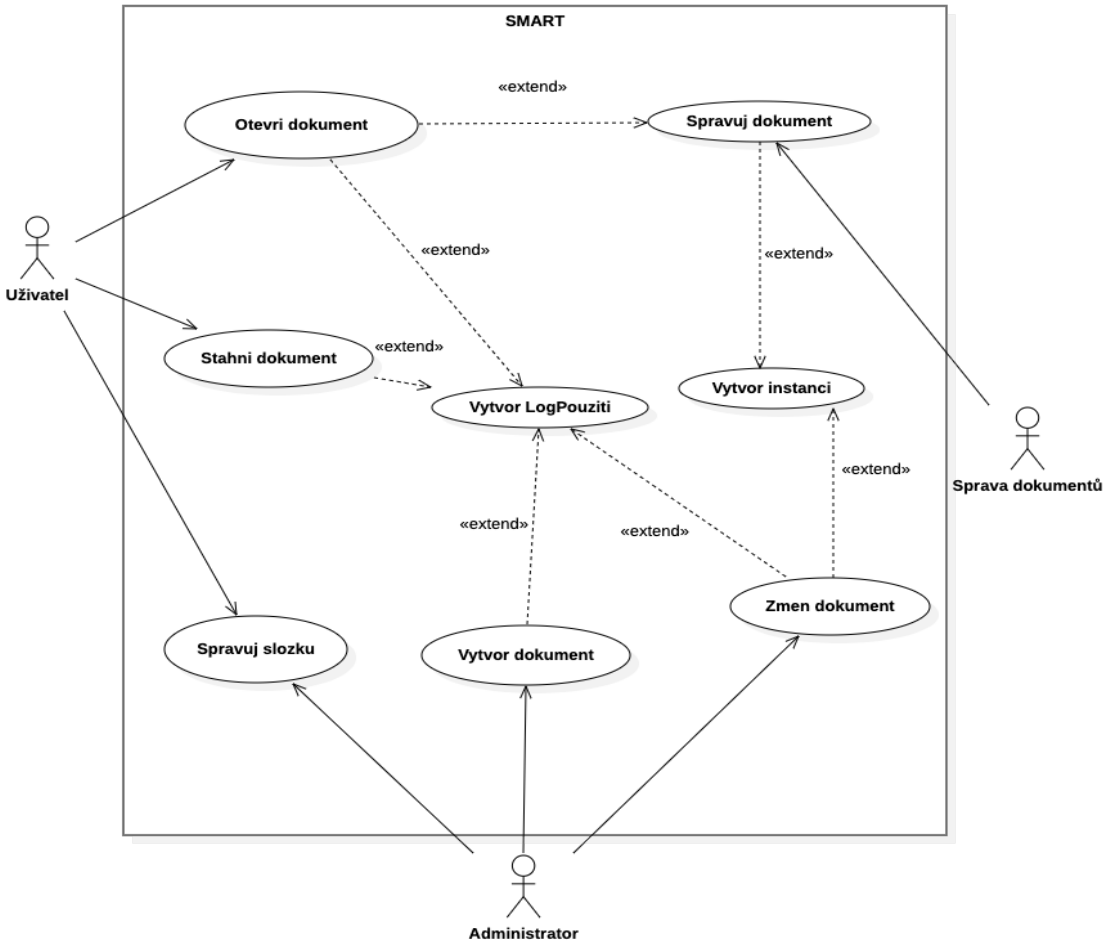
4.3.3 Model jednání

Model jednání také známý jako model interakcí je v UML graficky reprezentován interakcí mezi komponentami v systému. Používá se k modelování komunikace a koordinace mezi objekty a k popisu toho, jak spolupracují k dosažení společného cíle. Model interakcí je kritickým aspektem procesu návrhu v UML, protože pomáhá identifikovat problémy a přizpůsobit celý návrh v rané fázi procesu vývoje. Model interakcí tvoří tři diagramy:

diagram případů užití, sekvenční diagramy a diagramy aktivity. Další podkapitoly budou věnovány právě těmto diagramům pro vybrané komponenty modelovaného systému.

4.3.3.1 Use Case diagram

Diagram případů užití zachycuje pohled na IS pomocí interakcí aktérů a funkcionalitami systému. Samotné případy užití se formulují na základě stanovených funkčních požadavků. Use Case diagram má za cíl popsat, co vše systém umí, ale ne to, jak to dělá. Tedy jedná se o nejobecnější pohled na navrhovaný systém. Podkladem pro jakoukoliv interakci se systémem je potřeba být do něj přihlášen. Následovat bude obrázek s diagramem případů užití, který je doplněn tabulkou s popisem jednotlivých případů užití, aktér Správa dokumentů představuje modul externího systému určeného ke správě dokumentu ke správě:



Obrázek 24 – Use Case diagram (Zdroj: Vlastní zpracování)

Další tabulka představuje seznam případů užití, které jsou uvedeny v diagramu případů užití spolu s popisem:

Případ užití	Popis
Otevři dokument	S tímto případem užití spolupracuje aktér Uživatel. Umožňuje uživatelům prohlédnout dokument a jeho instance. Také následně pomocí připojeného případu užití *„Spravuj dokument“* ho může spravovat. Rovněž při otevření se generuje log, který je představený pomocí *„Vytvoř LogPoužiti“*, následně je uložen do databáze.
Stáhni dokument	Případ užití, se kterým interaguje **Uživatel**. Při provedení vidí uživatel seznam instancí a při stahování se vyvolává *„Vytvoř LogPoužiti“*, který uloží tuto aktivitu do databáze.
Vytvoř LogPouziti	Případ užití probíhající při otevření nebo stahování dokumentu, ukládá záznam o události do databáze.
Spravuj složku	Daný případ užití představuje funkcionalitu spojenou se správou složek, tedy úpravou obsahu, zakládáním nových a mazáním existujících. Při těchto událostech dochází ke spuštění *„Vytvoř LogPouziti“*.
Změň dokument	Tento případ užití znamená správu souboru u dokumentů ke stahování. Na požádaní uživatelů administrátor odpovídá za změny. Při změně se vytvoří instance, tedy popis provedené změny a verze dokumentu po změně, kterou také lze následně prohlédnout a zároveň se ukládá log o této události.
Spravuj dokument	S tímto případem užití spolupracuje aktér **Správa dokumentu**, modul externího systému, který umožňuje správu dokumentů uživatelům. Až změna je provedena, pak bude vytvořena instance, která popisuje údaje změny a uchovává verze dokumentu po změně.
Vytvoř instanci	Představuje případ užití, který je vyvolán, až byl změněn obsah dokumentu.
Vytvoř dokument	Případ užití, který označuje tvorbu dokumentu aktérem **Administrátor**.

Tabulka 5 – Seznam případů užití a jejich popis (Zdroj: Vlastní zpracování)

4.3.3.2 Sekvenční modely

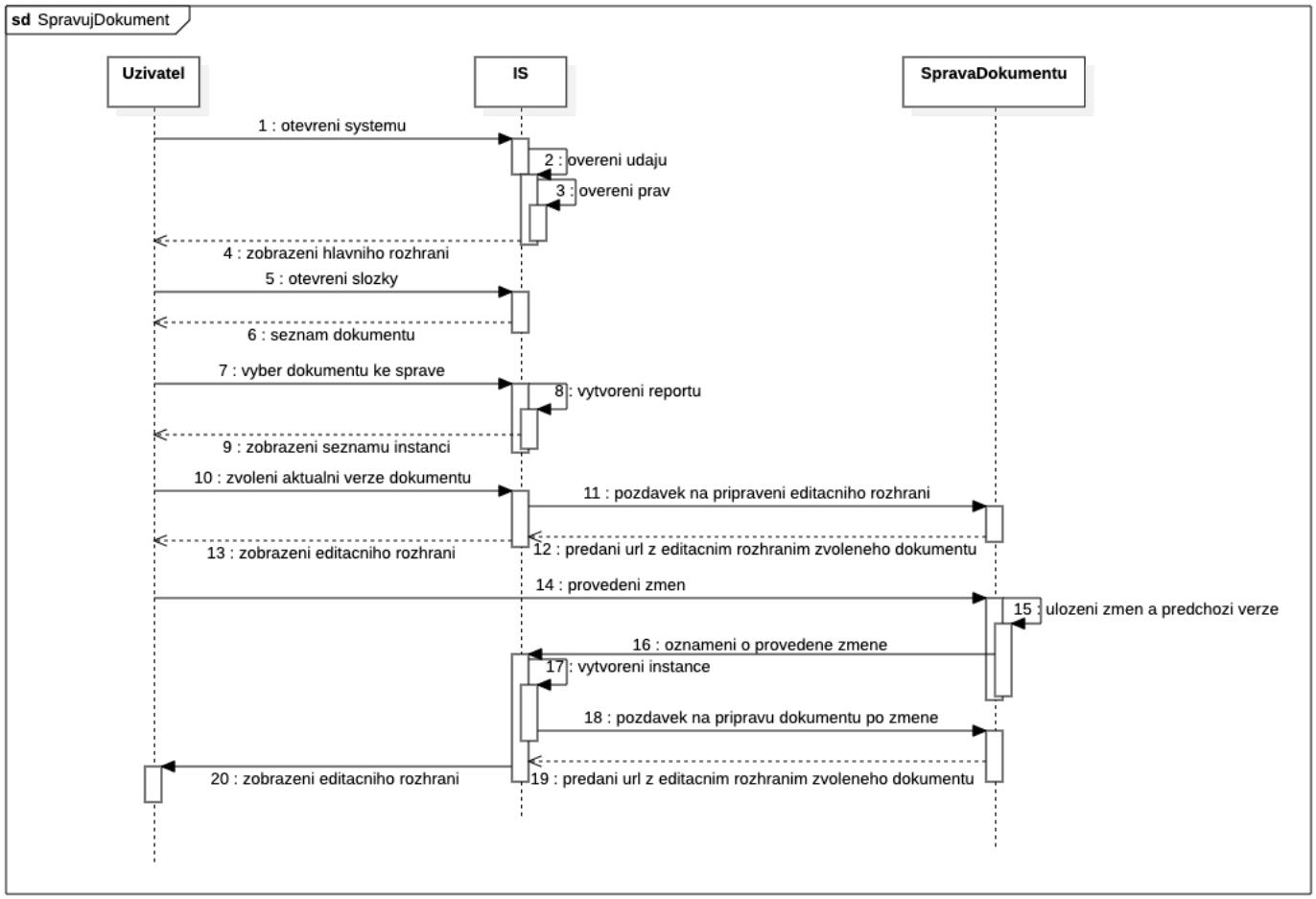
Sekvenční model je tvořen scénáři a sekvenčními diagramy, nebo také scénáři činností. Scénář představuje sekvenci událostí nebo akcí, které popisují konkrétní chování nebo tok aktivit v systému. Je to konkrétní příklad realizace případu užití, který ukazuje, jak spolu objekty v průběhu času interagují, aby splnily konkrétní úkol. Scénář obvykle obsahuje sadu interakcí mezi objekty, z nichž každá sada je reprezentována zprávou v sekvenčním diagramu. Dále budou uvedeny vybrané sekvenční modely.

4.3.3.2.1 Spravování dokumentu uživatelem

Případ užití: Spravování dokumentu	
Vstupní podmínky	Uživatel není přihlášen v systému. Uživatel má práva přístupu k systému. Uživatel má přiřazenou alespoň jednu neprázdnou složku s dokumentem ke správě. Dokument má instance.
Hlavní scénář	1. Uživatel otevře systém. 2. Systém ověří přihlašovací údaje uživatele. 3. Systém ověří práva uživatele 4. Systém zobrazí hlavní rozhrání. 5. Uživatel otevře složku. 6. Systém zobrazí seznam dokumentů. 7. Uživatel otevře dokument ke správě. 8. Systém vytvoří nový log použití. 9. Systém zobrazí seznam instancí. 10. Uživatel zvolí aktuální verzi. 11. Systém pošle požadavek na připravení editačního rozhraní do SpravyDokumentu. 12. SpravaDokumentu posila odkaz na připravené rozhraní. 13. Systém zobrazí editační rozhraní. 14. Uživatel mění údaje v dokumentu.

	15. SpravaDokumentu ukládá změny a verze dokumentu před změnou. 16. SpravaDokumentu oznamuje o provedené změně a posílá potřebné informace. 17. Systém ukládá změny a vytváří novou instanci dokumentu. 18. Systém posílá požadavek na přípravu dokumentu po změně. 19. SpravaDokumentu posílá odkaz na připravené rozhraní. 20. Uživatel vidí dokument po změně.
Výstupní podmínky	DokumentKeSprave byl změněn. Vytvoření logu použití. Vytvoření instance.
Alternativní scénář	Neplatné údaje k přihlášení. Nejsou dostatečná práva. SpravaDokumentu je nedostupná.

Tabulka 6 – Scénář pro případ užití spravuj dokument (Zdroj: Vlastní zpracování)



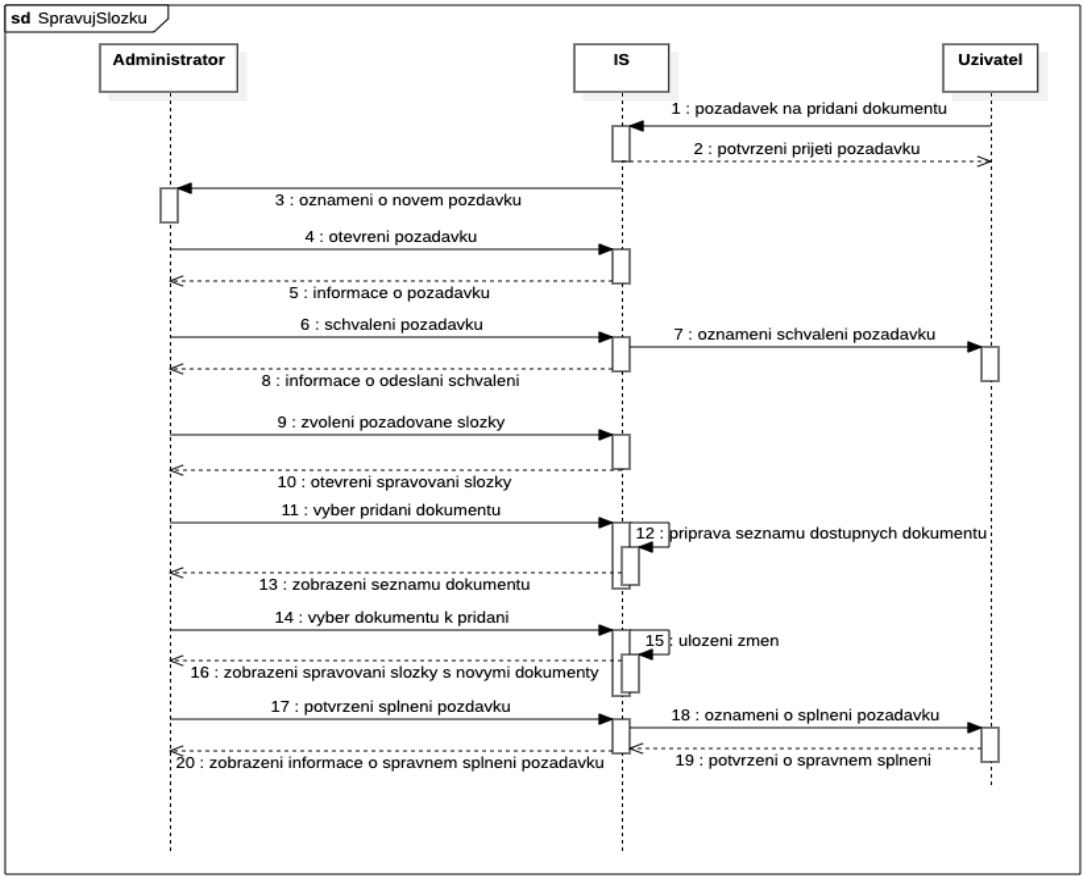
Obrázek 25 – Sekvenční diagram Spravuj dokument (Zdroj: Vlastní zpracování)

4.3.3.2.2 Přidání dokumentů do složky

Případ užití: Spravování Složku	
Vstupní podmínky	Administrátor je přihlášen v systému. Uživatel je přihlášen v systému. Potřebné dokumenty jsou v systému.
Hlavní scénář	1. Uživatel založí požadavek na přidání dokumentu. 2. Systém pošle potvrzení o přijetí požadavku. 3. Systém pošle oznámení o novém požadavku na administrátory. 4. Administrátor otevře požadavek. 5. Systém zobrazí informace o požadavku. 6. Administrátor schválí požadavek. 7. Systém pošle oznámení o schválení požadavku uživateli.

	8. Systém informuje administrátora o odeslání schválení. 9. Administrátor volí potřebnou složku. 10. Systém připraví rozhrání pro spravování složky. 11. Administrátor volí přidání dokumentů. 12. Systém připraví seznam dostupných dokumentů. 13. Systém zobrazuje dostupné dokumenty administrátorovi. 14. Administrátor vybírá dokumenty k přidání do složky. 15. Systém ukládá změny. 16. Systém připraví rozhraní pro spravování složky s novými dokumenty. 17. Administrátor potvrzuje splnění požadavku. 18. Systém odesílá oznámení uživateli. 19. Uživatel potvrzuje správné splnění požadavku. 20. Systém informuje administrátora o úspěšném vyřízení požadavku.
Výstupní podmínky	Úspěšné splnění požadavku o vložení nových dokumentů do složky.
Alternativní scénář	Neúspěšné načtení seznamu dokumentů. Požadavek nebyl splněn správně.

Tabulka 7 – Scénář pro případ užití Spravuj složku (Zdroj: Vlastní zpracování)



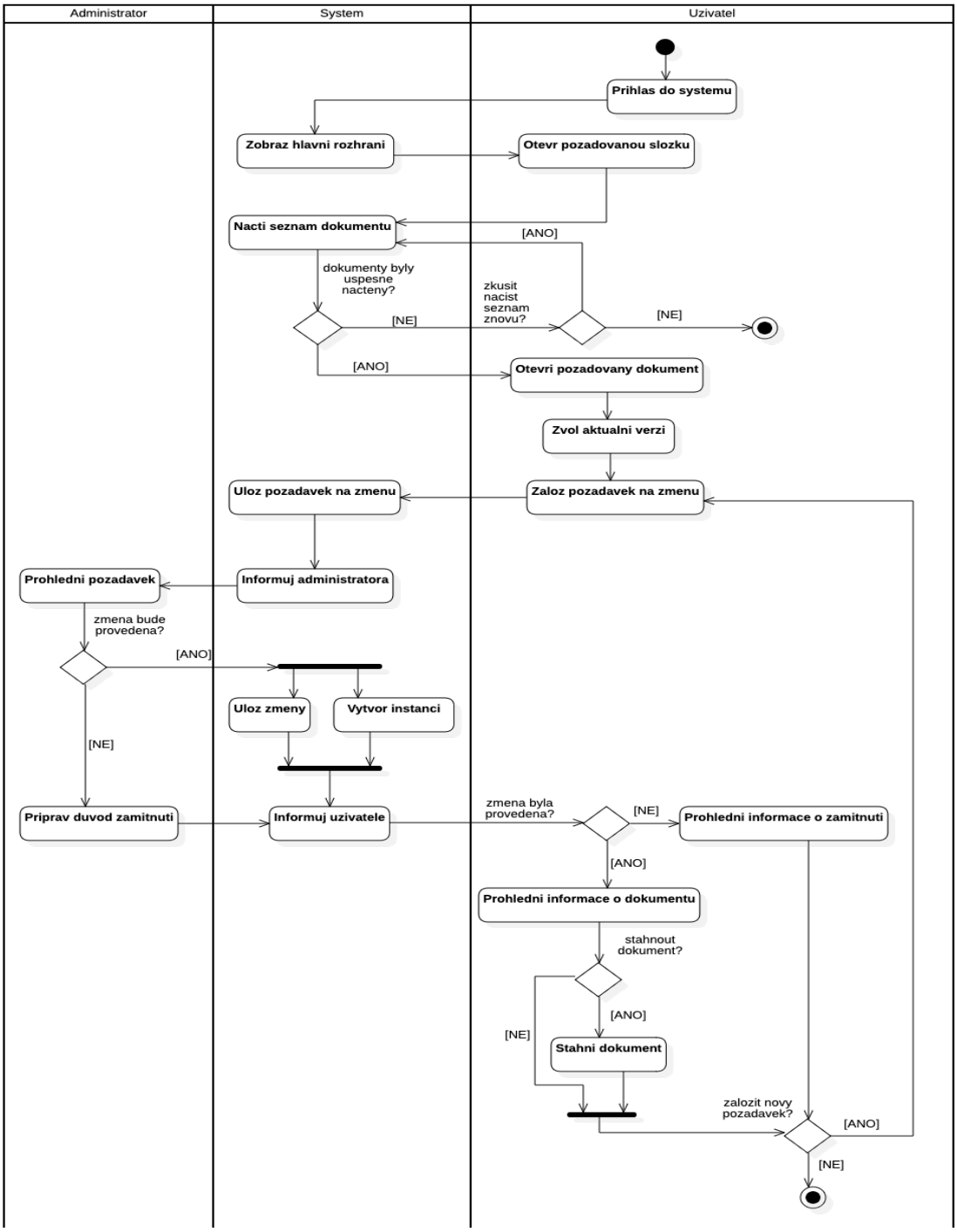
Obrázek 26 – Sekvenční diagram Spravuj složku (Zdroj: Vlastní zpracování)

Scénáře činností uvedené v této části jsou ty nejdůležitější pro porozumění interakcí probíhajících v systému. Obvykle se uvádí, že pro každý případ užití by měl být uveden alespoň jeden scénář, avšak ostatní případy podobné těm, které byly zmíněny nebo některé již byly i zahrnuty, např. „přihlas do systému“, „vytvoř instanci“ a „vytvoř report“.

4.3.3.3 Diagram aktivit

Diagram aktivit je užitečný nástroj pro modelování toku aktivit v rámci systému. Diagram aktivit je grafické znázornění tzv. workflow, které ukazuje posloupnost událostí a rozhodnutí, která jsou součástí procesu. Pomocí diagramů aktivit lze lépe porozumět aktivitám a procesům zapojených do informačního systému, což dále umožňuje identifikovat potenciální úzká místa a oblasti pro zlepšení.

Následující obrázek bude představovat diagram aktivit modelující proces změny dokumentu ke stažení na požadavek od uživatele:



Obrázek 27 – Diagram aktivit změna dokumentu ke stažení (Zdroj: Vlastní zpracování)

5 Výsledky a diskuse

Hlavním výsledkem této diplomové práce je návrh informačního systému v UML. Výchozím bodem pro rozhodnutí o zavedení nového systému je ztráta jeho aktuálnosti. Do toho se promítá nejen design a funkce poskytované IS, ale také bezpečnost, tedy výskyt kritických zranitelností zastaralých modulů ve zdrojovém kódu. Pokud podniku chci nejen dodávat nejlepší a nejmodernější uživatelské zkušenosti při práci se svým informačním systémem, ale i mít jistotu, že žádná data neuniknou, je potřeba vždy pravidelně aktualizovat tento systém. Právě k těmto účelům slouží výsledný návrh diplomové práce.

Pro navržení nového IS byla provedena analýza stávajícího systému. Také bylo využito pohovorů s uživateli, vedením týmu a vedoucím programátorem pro sestavení seznamu požadavků na systém nový.

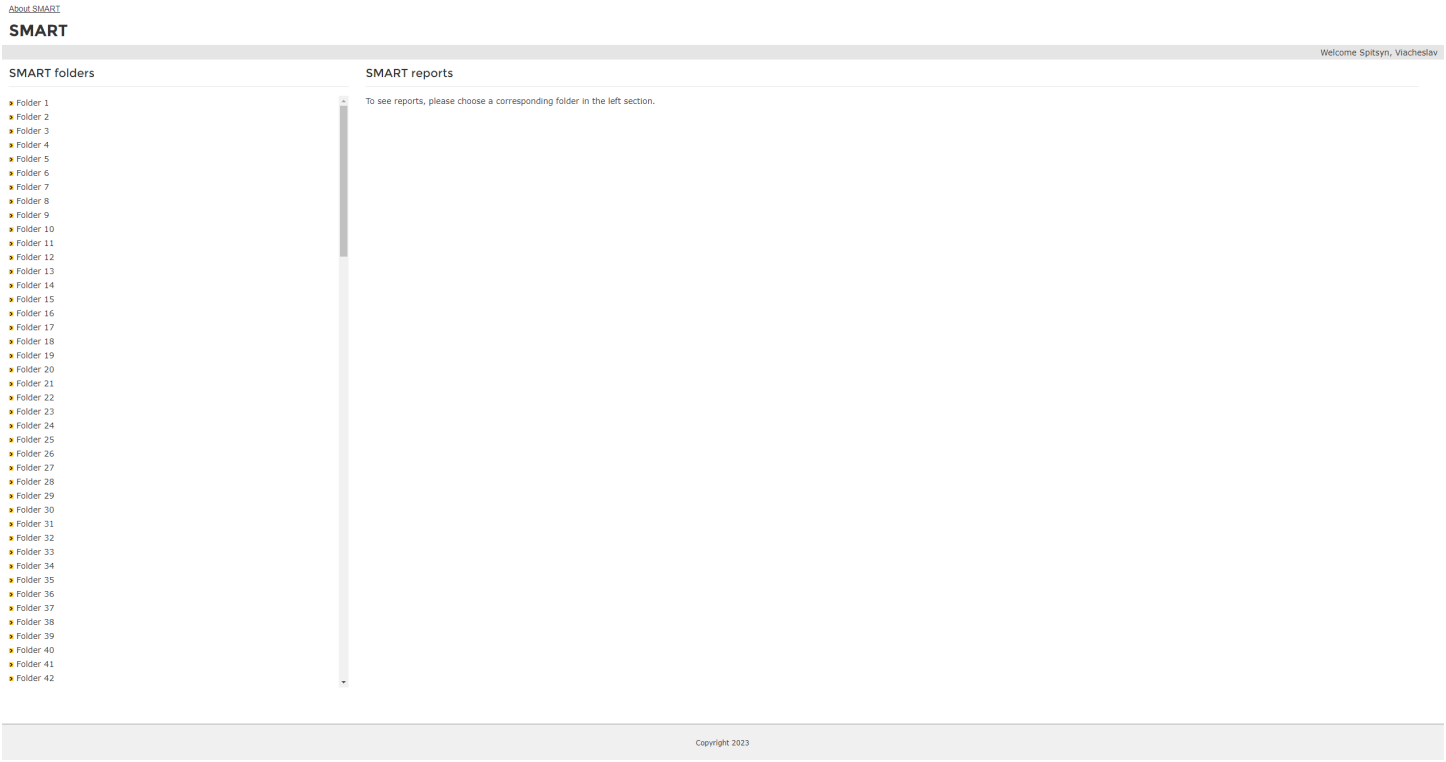
Výsledek je návrh informačního systému v UML, který by měl sloužit jako podklad pro následnou realizaci. Zároveň by mohl být použit jako část projektové dokumentace, jelikož zahrnuje nejpodstatnější funkce. Lze tedy říct, že navržený systém splňuje všechny na něj kladené funkční požadavky.

Realizace informačního systému podle návrhu zpracovávaného v rámci dané diplomové práce by měla zajistit splnění i nefunkčních požadavků. Jeden z nich je využití aktuální verze logovacího modulu log4j, který byl označen jako kritická zranitelnost, avšak se dá zcela přejít na jiný, například slf4j, který je taktéž známým nástrojem pro logování v programovacím jazyce Java. Obecně autor doporučuje z vlastní zkušenosti postavit nový systém na MVC architektuře, tj. rozdělení aplikace do tří propojených části, které umožní snadnější orientaci a následnou správu každé části systému zvlášť. MVC předpokládá abstrakce toho, co vidí uživatel od logiky, která je v tom skrytá. Taktéž stojí za zmínku, že framework Spring, který je zmíněný v nefunkčních požadavcích, obsahuje implementaci MVC. Co se týče designu a výsledného vzhledu IS, banka vede speciální stránky obsahující dokumentaci a potřebné zdroje pro vývoj programátory. Dokumentace by měla sloužit jako podklad pro realizaci uživatelského rozhrání.

5.1 Aktuální stav

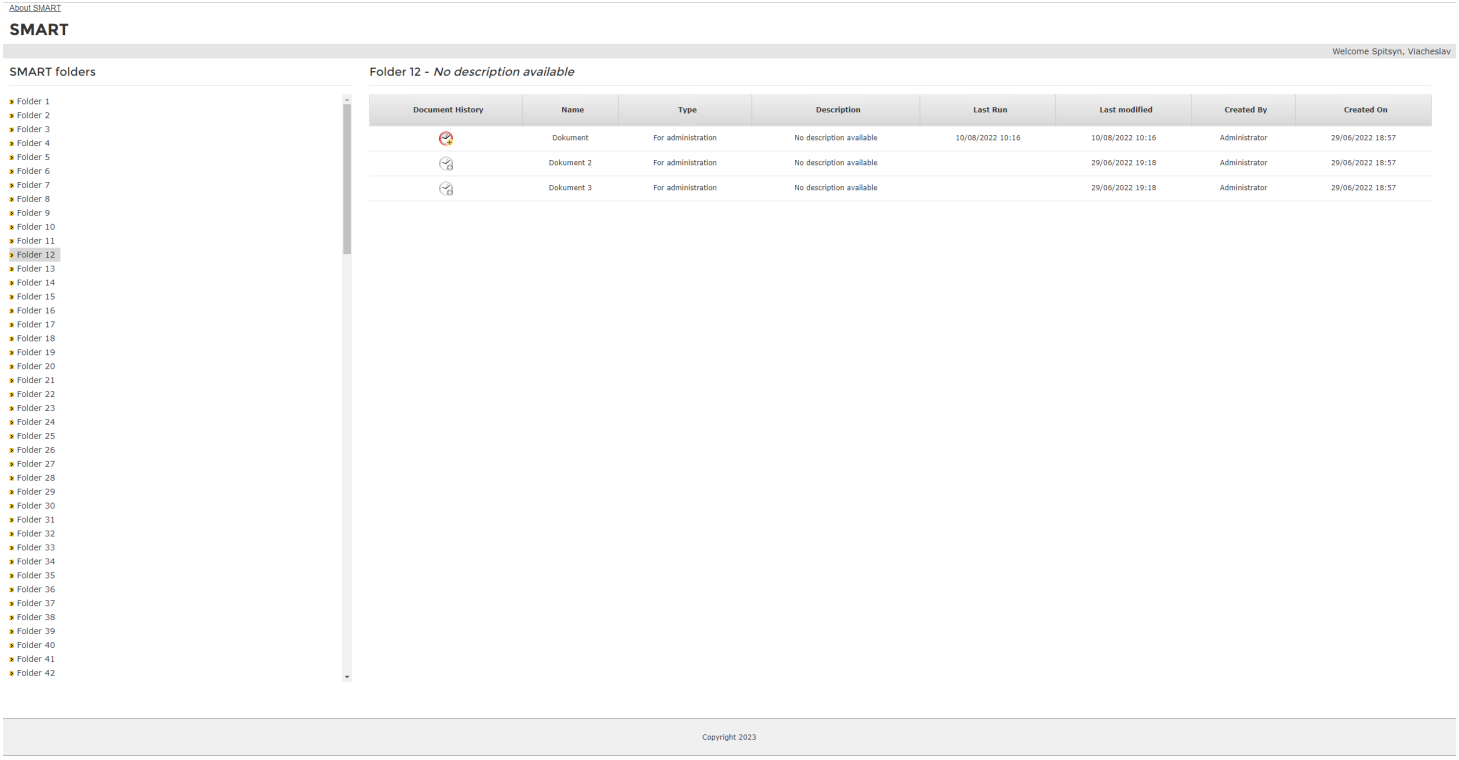
Autor diplomové práce se zabývá vývojem na základě zpracovaného návrhu v jazyce UML. Zatím všechny kladené požadavky na nový informační systém nebyly splněny. Informační systém ještě neumožňuje správu dokumentů ke stažení a zakládání požadavků na spravování složek v rámci rozhraní dostupného pro uživatele. Aktuálně toto probíhá přes emailovou komunikaci. Tyto chybějící funkce jsou ve fázi aktivního vývoje a brzy se plánují realizovat. Důvodem k odložení zavedení všech požadovaných funkcí najednou je potřeba, co nejrychleji se zbavit kritických zranitelností a také možnost testování pro odstranění objevených chyb a defektů ještě během vývoje.

Následující obrázek představuje uživatelské rozhraní po otevření informačního systému. Z bezpečnostních důvodů byl logotyp banky odstraněn a data vyměněna za fiktivní.



Obrázek 28 – Uživatelské rozhraní po otevření systému (Zdroj: Vlastní zpracování)

Seznam vlevo představuje seznam složek dostupných uživateli. Po zvolení konkrétní složky (kliknutím na ni) uživatel vidí tabulku s dokumenty, které jsou součástí dané složky. Toto je vidět na dalším obrázku:



Obrázek 29 – Zobrazení dokumentů po otevření složky (Zdroj: Vlastní zpracování)

Aktivní tlačítko pod sloupcem s názvem „Document History“ indukuje o přítomnosti instanci u daného dokumentu. Po kliknutí na ikonku se otevře modální okno se seznamem instanci.

Toto je k vidění na následujícím obrázku:

Obrázek 30 – Seznam instanci (Zdroj: Vlastní zpracování)

Zatím uživatelé kladně hodnotí nový design uživatelského rozhrání a možnost spravování dokumentu v rámci jedné aplikace, nemusí znát ID jednotlivých dokumentů a jít do externího systému. Práce je proto pohodlnější. Vedení oddělení, které má na starosti rozvoj a údržbu IS, se už nemusí obávat za objevené kritické závislosti. Je samozřejmé, že v průběhu času se můžou vyskytnout i nové zranitelnosti, ale s modernější infrastrukturou by následující modernizace měla být jednodušší. Pro vedoucího programátora a autora dané práce, který také odpovídá za technickou realizaci a vývoj, je teď k dispozici díky většímu množství logů vetší množství prostředků pro odhalení chyb a celkový plynulý chod IS.

6 Závěr

Návrh informačního systému v UML je složitý proces, který vyžaduje důkladné pochopení jak základních principů tohoto modelovacího jazyka, tak i požadavků na tento systém kladených.

Cílem této diplomové práce byl návrh informačního systému v UML, který by se měl stát náhradou pro systém stávající, tedy interní bankovní aplikace pro prohlížení složek s dokumenty buď pro následnou správou nebo stahování uživatelem.

Pro splnění cíle diplomové práce byla prostudována oblast dané problematiky ve formě literární rešerše knižních a webových zdrojů. Na začátku byl představen úvod do informačních systémů, tedy základní pojmy a rozdělení spolu s popisem. Následně byly popsány fáze životního cyklu IS a metody v jeho vývoji. Dále autor uvedl druhy přístupů k analýze a návrhu informačního systému, konkrétně strukturovaný a objektově orientovaný přístup. Také se seznámil softwarovými nástroji CASE, jež podporují práci při zpracování návrhu. Poslední část teoretického zakladu byla věnována modelovacímu jazyku UML. Zde byl tento jazyk prozkoumán spolu se základními vlastnostmi a jednotlivými nástroji. Diagramy v UML tvoří jádro jazyka pro návrh informačních systémů. Byly představeny diagramy spadající do tří modelů, které ukazují navrhovaný systém z různých pohledů pro lepší představení chodu, prvků a vazeb mezi nimi, tedy to, jak by měl ve výsledku IS fungovat.

Vlastní část práce byla soustředěna na návrh informačního systému v UML. Pro dosažení hlavního cíle bylo potřeba analyzovat stávající systém, jelikož to by měl být základ pro systém nový. Zatímco byla zpracována analýza systému stávajícího, byly sebrány funkční a nefunkční požadavky na navrhovaný systém. Požadavky se sbíraly pomocí pohovorů s uživateli stávajícího systému, vedením týmu a vedoucím programátorem. Návrh byl zpracován pomocí tří modelů a diagramů do nich spadajících. Jmenovitě modelu tříd stavovému modelu a modelu jednání. Model tříd udává statickou stránku systému a je v práci představen pomocí diagramu tříd, který ukazuje jednotlivé třídy a vazby mezi nimi. Stavový model slouží pro představení chování objektů nebo systému v průběhu času, a je tvořen stavovými diagramy. Poslední zmíněný model je modelem jednání neboli interakcí, a je

grafickou reprezentací interakcí mezi komponentami v systému, který byl reprezentován pomocí diagramu případů užití, sekvenčních modelů a diagramu aktivity.

V poslední části diplomové práce jsou výsledky a diskuse. Tato část byla věnována dosaženému výsledku, tedy návrhu informačního systému v UML a diskusi nad tím, jak by mohla vypadat následná realizace z pohledu autora a aktuální stav, ve kterém se navrhovaný systém nachází.

Celkově práce zdůraznila důležitost analýzy a návrhu při vývoji informačního systému. Také uvádí klíčovost včasné modernizace IS. To by mělo nejen zvýšit bezpečnost a zlepšit každodenní práci řádných uživatelů, ale zároveň i zvyšuje postavení společnosti vůči okolí.

7 Seznam použitých zdrojů

[1] KLEMENT, Milan. Teorie systémů – úvod do teorie informačních systémů. Olomouc: Univerzita Palackého v Olomouci, 2022. ISBN 978-80-244-6109-9.

[2] VANĚK, Jiří. Informační technologie I: vybrané kapitoly. Praha: Credit, 2004. ISBN 978-80-213-1122-0.

[3] GÁLA, Libor, Jan POUR a Zuzana ŠEDIVÁ. Podniková informatika: počítačové aplikace v podnikové a mezipodnikové praxi. 3., aktualizované vydání. Praha: Grada Publishing, 2015. Management v informační společnosti. ISBN 978-80-247-5457-4.

[4] SCHEJBAL, Ctirad, Vladimír HOMOLA a František STANĚK. Geoinformatika. Košice: Pont, 2004. ISBN 80-967611-8-8.

[5] VYMĚTAL, Dominik. Informační systémy v podnicích: teorie a praxe projektování. Praha: Grada, 2009. Průvodce (Grada). ISBN 978-80-247-3046-2.

[6] MOLNÁR, Zdeněk. Podnikové informační systémy. Vyd. 2., přeprac. V Praze: České vysoké učení technické, 2009. ISBN 978-80-010-4380-6.

[7] BRUCKNER, Tomáš. Tvorba informačních systémů: principy, metodiky, architektury. Praha: Grada, 2012. Management v informační společnosti. ISBN 978-80-247-4153-6.

[8] DANEL, Roman. Informační systémy – elektronická skripta [online]. 2011 [cit. 2023-01-13]. Dostupné z: http://homel.vsb.cz/~dan11/is_skripta/IS%202011%20-%20TPS-MIS-EIS.pdf

[9] PALATKOVÁ, Monika. Marketingový management destinací: strategický a taktický marketing destinace turismu, systém marketingového řízení destinace a jeho financování, řízení kvality v destinaci a informační systém destinace. Praha: Grada, 2011. ISBN 978-80-247-3749-2.

[10] VOŘÍŠEK, Jiří. Strategické řízení informačního systému a systémová integrace. Praha: Management Press, 1997. ISBN 80-85943-40-9.

[11] GRiT: Co je to EDI? [online]. 2023 [cit. 2023-01-13]. Dostupné z: https://www.grit.eu/co-je-edi

[12] KAJZAR, Dušan a Ivan POLÁŠEK. Projektování informačních systémů I: strukturovaný a objektový přístup. V Opavě: Slezská univerzita, Filozoficko-přírodovědecká fakulta, Ústav informatiky, 2003. ISBN 80-7248-214-9.

[13] ŠIMONOVÁ, Stanislava, Renáta MYŠKOVÁ a Pavel JIRAVA. Projektování informačních systémů – UML, procesní řízení. 2. vyd. Pardubice: Univerzita Pardubice, Ekonomicko-správní fakulta, 2008. ISBN 978-80-7395-131-3.

[14] VRANA, Ivan a Karel RICHTA. Zásady a postupy zavádění podnikových informačních systémů: praktická příručka pro podnikové manažery. Praha: Grada, 2005. Management v informační společnosti. ISBN 80-247-1103-6.

[15] ROYCE, Winston. Managing the Development of Large Software Systems [online]. Proceedings of IEEE WESCON, 1970 [cit. 2023-01-15]. Dostupné z: http://www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf

[16] Vodopádový model: Vodopádový model životního cyklu software (The waterfall Life cycle). Testování softwaru [online]. [cit. 2023-01-15]. Dostupné z: http://testovanisoftwaru.cz/manualni-testovani/modely-zivotniho-cyklu-softwaru/vodopadovy-model/

[17] BOEHM, Barry. A Spiral Model of Software Development and Enhancement [online]. TRW Defence Systems Group, 1986 [cit. 2023-01-15]. ISSN 0018-9162. Dostupné z: https://classes.cs.uoregon.edu/07W/cis422/readings/SpiralModel.pdf

[18] Testování softwaru: Spirálový model [online]. [cit. 2023-01-16]. Dostupné z: http://testovanisoftwaru.cz/manualni-testovani/modely-zivotniho-cyklu-softwaru/spiralovy-model/

[19] BUCHALCEVOVÁ, Alena. Metodiky vývoje a údržby informačních systémů: kategorizace, agilní metodiky, vzory pro návrh metodiky. Praha: Grada, 2005. Management v informační společnosti. ISBN 80-247-1075-7.

[20] BECK, Kent. Extreme Programming Explained: Embrace Change. 2. the University of Michigan: Addison-Wesley Professional, 2000. ISBN 9780201616415.

[21] PALMER, Stephen R. a John M. FELSING. A Practical Guide to Feature-driven Development. Upper Saddle River NJ: Prentice Hall PTR, 2002. ISBN 9780130676153.

[22] POLÁK, Jiří, Antonín CARDA a Vojtěch MERUNKA. Umění systémového návrhu: objektově orientovaná tvorba informačních systémů pomocí původní metody BORM. Praha: Grada, 2003. Management v informační společnosti. ISBN 80-247-0424-2.

[23] BLAHA, M., RUMBAUGH, J. Object-oriented modeling and design with UML. Upper Saddle River, NJ: Pearson Education, 2005. ISBN 0130159204.

[24] ŽÁČEK, Jaroslav. Softwarové inženýrství [online]. Ostrava, 2014 [cit. 2023-01-22]. Dostupné z: https://web.osu.cz/~Zacek/sweng/skripta_sweng.pdf

[25] ŽÁČEK, Jaroslav. CASE nástroje [online]. [cit. 2023-01-22]. Dostupné z: https://web.osu.cz/~Zacek/sweng/08.pdf

[26] FOWLER, Martin. UML distilled: A Brief Guide to the Standard Object Modeling Language. 3. Addison-Wesley Professional, 2003. ISBN 0321193687.

[27] BOOCH, Grady, James RUMBAUGH a Ivar JACOBSON. The unified modeling language user guide. 2nd ed. Upper Saddle River: Addison-Wesley, 2005. ISBN 0321267974.

[28] OMG Unified Modeling Language (OMG UML): Version 2.5.1 [online]. 2017 [cit. 2023-01-25]. Dostupné z: https://www.omg.org/spec/UML/2.5.1/PDF

[29] VRANA, Ivan. Projektování informačních systémů s UML. V Praze: Česká zemědělská univerzita, Provozně ekonomická fakulta, 2008. ISBN 978-80-213-1817-5.

[30] UML Use Case Diagram [online]. [cit. 2023-01-31]. Dostupné z: https://www.javatpoint.com/uml-use-case-diagram

[31] BELL, Donald. IBM. Explore the UML sequence diagram: Interactions between objects in the sequential order that those interactions occur [online]. 2004 [cit. 2023-01-31]. Dostupné z: https://developer.ibm.com/articles/the-sequence-diagram/

8 Seznam obrázků, tabulek a zkratek

8.1 Seznam obrázků

8.2 Seznam tabulek

8.3 Seznam použitých zkratek

IS	*Informační systém*
UML	*Unified Modelling Language*
MIS	*Management Information System*
CASE	*Computer Aided Software Engineering*
OMG	*Object Management Group*
EIS	*Executive Information System*
TPS	*Transaction Processing System*
EDI	*Electronic Data Interchange*
OIS	*Office Information Systém*
OOP	*Objektově orientováný přístup*
OMT	*Object Modelling Technique*
OOSE	*Object-Oriented Software Engineering*
FDD	*Feature-Driven Development*
MVC	*Model-View-Controller*