Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Návrh informačního systému v UML

Bc. Jan Karel

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Jan Karel

Systémové inženýrství a informatika
Informatika

Název práce

Návrh informačního systému v UML

Název anglicky

Information System Design in UML

Cíle práce

Diplomová práce je tématicky zaměřena na návrh a projektování informačního systému. Hlavním cílem této práce je za použití jazyka UML navrhnout diagram tříd, který bude reprezentovat statickou strukturu systému, znázorňovat datové struktury, operace u objektů a základní souvislosti mezi objekty. Dílčím cílem je převod jednotlivých částí diagramu do objektově orientovaného programovacího jazyku (Java).

Metodika

Metodika řešené problematiky diplomové práce vychází ze studia a analýzy odborných informačních zdrojů. Na základě získaných teoretických poznatků dojde k vytvoření návrhu diagramu tříd tak, aby došlo k zachování klíčových prvků systému a celkové smysluplnosti. V návaznosti na vytvořený diagram dojde z programátorské hlediska k přepsání do programovacího jazyku (Java). Na závěr dojde k vyhodnocení celkového diagramu tříd a posouzení programátorské transformace z jazyka UML.

Doporučený rozsah práce

60-80

Klíčová slova

UML, diagram tříd, Java, informační systém, objektově orientovaný přístup

---

Doporučené zdroje informací

AMBLER, S W. *The elements of UML 2.0 style.* Cambridge: Cambridge University Press, 2005. ISBN 0-521-61678-6.

ARLOW, J. – NEUSTADT, I. *UML a unifikovaný proces vývoje aplikací : průvodce analýzou a návrhem objektově orientovaného softwaru.* Brno: Computer Press, 2003. ISBN 80-7226-947-.

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

ECKEL, B. *Myslíme v jazyku Java : knihovna programátora.* Praha: Grada, 2001. ISBN 80-247-9010-6.

FLANAGAN, D. – KRÁSENSKÝ, D. *Programování v jazyce JAVA.* Praha: Computer Press, 1997. ISBN 80-85896-78-8.

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

2020/21 LS – PEF

Vedoucí práce

Ing. Jan Tyrychtr, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 19. 11. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 11. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 28. 03. 2021

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 30.3.2021 _____

Poděkování

Děkuji touto cestou vedoucímu diplomové práce Ing. Janu Tyrychtrovi, Ph.D., za trpělivost, ochotu a cenné rady při vypracování diplomové práce. Mé poděkování patří též Mgr. Václavě Čápové za pomoc při gramatické kontrole práce. Velké poděkování patří i rodině a blízkým přátelům za neutuchající podporu během studia.

Návrh informačního systému v UML

Abstrakt

Diplomová práce se zabývá návrhem informačního systému v UML pro vybraný podnik. Jedním z prvních kroků při tvorbě návrhu je definovat funkční a nefunkční požadavky. Na základě definovaných požadavků lze následně přistoupit k návrhu statické struktury systému. K tomuto typu návrhu se nejčastěji používá diagram tříd, který nejenže definuje jednotlivé třídy s atributy a operacemi, ale také zajišťuje jejich vzájemnou komunikaci skrze vztahy. Výsledný diagram je mimo jiné charakteristický tím, že se v identifikátorech již nevyskytuje diakritika a atributy mají datové typy specifické pro daný jazyk. Navržený diagram tříd tedy lze transformovat do příslušného programovacího jazyka, kterým je v tomto případě Java. Výsledkem by měl být navržený diagram tříd, který lze pomocí jednotlivých specifik převést do programovacího jazyka.

Klíčová slova: UML, diagram tříd, Java, informační systém, objektově orientovaný přístup.

Information System Design in UML

Abstract

This diploma thesis describes the design of an information system in UML for selected business. One of the first steps in creating a design is to define functional and non-functional requirements. Based on the defined requirements, it is then possible to proceed to the design of the static structure of the system. For this type of design, a class diagram is most often used, which not only defines individual class with attributes and operations, but also ensures their mutual communication through relationships. The resulting diagram is characterized, among other things, by the fact that the diacritics no longer appear in the identifiers and the attributes have data types specific to the given language. The proposed diagram can be transformed into the appropriate programming language, which in this case is Java. The result should be a designed class diagram, which can be converted into a programming language using individual specifics.

Keywords: UML, class diagram, Java, information system, object-oriented programming.

Obsah

Seznam obrázků

Seznam tabulek

Seznam kódu

Seznam použitých zkratek

IS – Information System

BI – Business Inteligence

EIS – Executive Information System

MIS – Management Information System

TPS – Transaction Processing System

EDI – Electronic Data Interchange

OIS – Office Information System

DSS – Decision Support System

ERP – Enterprise Resourse Planning

CRM – Customer Relationship Management

SCM – Supplier Chain Management

URL – Uniform Resource Locator

OLTP – Online Transaction Processing

APS – Advanced Planning Scheduling

KCRM – Knowledge-enabled Customer Relationship Management

XP – Extreme Programming

FDD – Feature-driven Development

JAD – Join Application Design

OOP – Object-oriented Programming

UDP – User Datagram Protocol

1 Úvod

Pojem informační systém se nedotýká pouze nadnárodních společností s dostatečně velkým kapitálem, které si implementaci mohou dovolit. S narůstajícím trendem po zpracování příslušných dat, jejich interpretování na informace a následné uchopení ve formě znalostí, které později mohou vyústit v moudrost, se stále více menších či středních podniků zabývá otázkou, jaký informační systém pro svůj podnik vybrat.

Navrhnutí a výsledné implementování informačního systému není mnohdy krátkodobou záležitostí, která by trvala několik týdnů, spíše se jedná o měsíce, či dokonce léta. Často na návrhu nepracuje pouze jeden člověk, ale hned soubor několika kvalifikovaných lidí. Samotný návrh se nejvíce opírá o požadavky zadavatele či uživatele, který bude se systémem pracovat. Jakmile zde dojde k nejasné interpretaci myšlenek, může to stát kromě vynaloženého času také nemalé peníze, což si menší podnik ne vždy může dovolit. Je proto potřebné pracovat již od začátku svědomitě a pečlivě.

Nedílnou součástí návrhu je zachycení statické struktury systému, kterou lze poměrně čitelně a efektivně vyjádřit pomocí jazyka UML, konkrétně pomocí diagramu tříd. Diagram tříd zpracovávaný většinou analytiky je ve výsledku poměrně jednoduše čitelný pro programátory, kteří následně mají za úkol převést diagram do zvoleného programovacího jazyka. Tato část je velmi důležitá, neboť z něčeho, co kdysi bývalo jen pomyslným návrhem na papíře, se v průběhu času stává něco hmatatelného a užitečného, což jistě nejeden podnik ocení.

Výsledkem je hotový a implementovaný informační systém, který by po úspěšném spuštění měl vybranému podniku přinést zásadní konkurenční výhodu, jež může vést nejen k upevnění pozice na trhu, ale také k dalšímu růstu a rozvoji.

2 Cíl práce a metodika

2.1 Cíl práce

Tématem diplomové práce je návrh a tvorba informačního systému. Cílem práce je sběr funkčních a nefunkčních požadavků, což lze považovat za nezbytnou součást návrhu systému. Na základě získaných požadavků dojde k navržení jednotlivých tříd, přičemž třídy budou seskupeny pro lepší orientaci do vybraných packages. Při návrhu bude kladen důraz na správnost a korektnost řešení. Výsledkem této části bude diagram tříd, který bude reprezentovat statickou strukturu systému, znázorňovat datové struktury, operace u objektů a základní souvislosti mezi objekty. Dílčím cílem bude převod diagramu do programovacího jazyka Java. Během transformace dojde k zachycení klíčových aspektů a technik, které spolu vzájemně souvisí.

2.2 Metodika

Metodika řešení problematiky vychází ze studia a analýzy odborných informačních zdrojů. Na základě získaných teoretických poznatků a vlastních zkušeností dojde k definování jednotlivých postupů u dílčích částí této diplomové práce. Sběr požadavků je nedílnou součástí návrhu informačního systému. Existuje celá řada technik, jak kvalitně a efektivně provést sběr požadavků, nicméně pro účely této práce bude použita technika interview. K návrhu a vhledu do vnitřního uspořádání systému bude využit jazyk UML, konkrétně packages a diagram tříd. Strukturální diagram bude vycházet z jeho základní struktury, kterou tvoří třídy, atributy, operace, vztahy či mechanismy rozšiřitelnosti. Použitým nástrojem bude Enterprise Architect od společnosti Sparx Systems.

K transformaci diagramu tříd do programovacího jazyka Java bude použit nástroj Apache NetBeans IDE. Ten obsahuje plugin easyUML. Tento nástroj poskytuje kromě funkce pro snadné vytváření diagramů v UML také generování kódu na základě tzv. reverzního inženýrství, které dokáže základní aspekty diagramu tříd přetransformovat do programovacího jazyka. Základními aspekty jsou myšleny vybrané relace a třídy, včetně jejich atributů a operací. Součástí transformace jsou též datové typy a viditelnost.

3 Teoretická východiska

Nežli bude přistoupeno k praktické části, je nutno nejprve specifikovat oblast informačních systémů, analytických a programovacích jazyků. Na základě toho dojde k lepšímu pochopení veškerých funkcionalit a jejich propojení.

3.1 Definice informačních systémů

V návaznosti na rozsáhlost a v některých případech i komplikovanost informačních systémů, není jednoduché stanovit přesnou a jedinou definici informačního systému. Pro účely a smysluplnost této práce jsou uvedeny tři vybrané definice. Informační systém organizace může být definován jako:

Informační systém je soubor lidí, technických prostředků a metod, zabezpečující sběr, přenos, zpracování, uchování dat, za účelem prezentace informací pro potřeby uživatelů činných v systémech řízení. [1]

Systém informačních a komunikačních technologií, dat a lidí, jehož cílem je efektivní podpora informačních, rozhodovacích a řídicích procesů na všech úrovních řízení organizace. [2]

Takový systém, kde se vazby mezi prvky systému a vazby s okolím realizují prostřednictvím dat a informací. [3]

3.2 Data, informace, znalost

Pokud chápeme současnou pozici informačních systémů, která svým neoddiskutovatelným přínosem mění informační evoluci, tak je nutné si uvědomit, že základem zvyšování komfortu uživatelů či jiných aspektů, je zlepšování srozumitelnosti a interpretace podnikových dat. Na základě toho lze mnohem snadněji data transformovat na informace, které jsou mnohdy považovány za hlavní zdroj podnikání, stejně jako kapitál či práce.

3.2.1 Data

Jak uvádí Sodomka a Klčová [4], data představují neodmyslitelný prvek podnikového informačního systému. Jsou nositeli zaznamenaných skutečností souvisejících s aktivitami podniku a zároveň jsou schopna přenosu, intepretace a zpracování.

Pour, Gála a Šedivá [5] uvádějí, že data lze rozdělit do tří skupin:

1) Data o společenských podmínkách podnikání – zahrnují veškeré poznatky o mikro a makrookolí organizace, jako jsou zaznamenané údaje o demografických, sociálních, ekonomických trendech společnosti, pracovní síle, dostupnosti materiálu, kapitálu a dalších faktorech ovlivňujících hodnototvorný řetězec firmy.
2) Data o trhu – tvoří zaznamenané skutečnosti o nabídce, poptávce, konkurenci a celkovém dění na trhu včetně očekávaných akvizic, tvorby strategických aliancí apod.
3) Interní data – jsou nositeli faktů umožňujících managementu „poznat svůj podnik" a správně reagovat na své okolí. Do této skupiny patří obchodní a finanční plány, predikce vývoje, data o podnikových zdrojích, jejich alokaci a omezeních, data nesoucí vnitřní normy, pravidla a procedury podniku.

3.2.2 **Informace**

Informace z hlediska podniku jsou rozdělovány na informace externí a informace interní. Externí informační zdroje jsou koncipovány pro široké využívání relativně rozsáhlými a nesourodými skupinami uživatelů. Získávání externích informací je z hlediska podniku časově i finančně náročné. Pro kategorii malých a středních podniků je kvalitní IS poskytující externí informace poměrně značnou ekonomickou zátěží. Interním zdrojem informací pro podnik jsou všechna data o stavu podniku, tj. zejména finanční ukazatele, stavy zásob, zaměstnanci. [6]

3.2.3 **Znalost**

Znalosti jsou založené především na interpretaci, zkušenostech, poznávání a porozumění. Dále jsou závislá na inteligenčních schopnostech a na schopnostech dávat si informace do souvislosti. Základem každé znalosti je schopnost tvorby či získání poznatků (informací), následné uspořádání, přenos, sdílení až po výsledné používání.

3.3 Význam informačních systémů

Informační systémy dnes podporují nejen všechny důležité podnikové funkce, jakými jsou například finance, personalistika, plánování, prodej, nákup, logistika včetně e-businessu a m-businessu. IS musí v současnosti umět držet krok s businessem a jeho potřebami – tj. například s různými podnikovými fúzemi a trvalými požadavky na podporu efektivnosti, flexibility a inovace důležitých podnikových procesů, produktů a služeb. [7]

Z výše uvedeného významu je patrné, že informační systémy jsou a v budoucnu jistě budou nedílnou součástí podniků a jejich podnikových procesů. Je to dáno jednak stále větší intenzitou a efektivností v podnikání, což vyžaduje zcela nové modely podnikání. V současné době již není tolik přihlíženo k uvedení systému do provozu, ale čím dále více k business přínosu pro celý podnik či jednotlivá oddělení.

3.4 Klasifikace informačních systémů

Každý podnik je charakteristický tím, že je rozdělen na několik organizačních úrovní. Ačkoli se tyto úrovně mohou tvářit jako samostatně fungující celky, opak je pravdou, neboť podnik by jen těžko mohl prosperovat, pokud by jedna z úrovní nefungovala. Z tohoto důvodu se zavádějí informační systémy pro celý podnik, kde každá úroveň sice přináší rozdílné informace, spolu ovšem tvoří kompletní strukturu podniku. Informační systém proto nemůže být nasazen jen na jednu organizační úroveň. Následující kategorie úrovní úzce korespondují se samotným fungováním podniku.

Obrázek 1 Informační pyramida podle organizačních úrovní podniku [4]

Provozní úroveň – požaduje zpracování informací týkajících se rutinní podnikové agendy, jako je realizace výrobních zakázek, nákupu a prodeje, příjmu plateb a výplat apod.

Informační systémy pokrývající provozní úroveň reagují na plnění každodenní činnosti a sledují tok transakcí napříč organizací. [4]

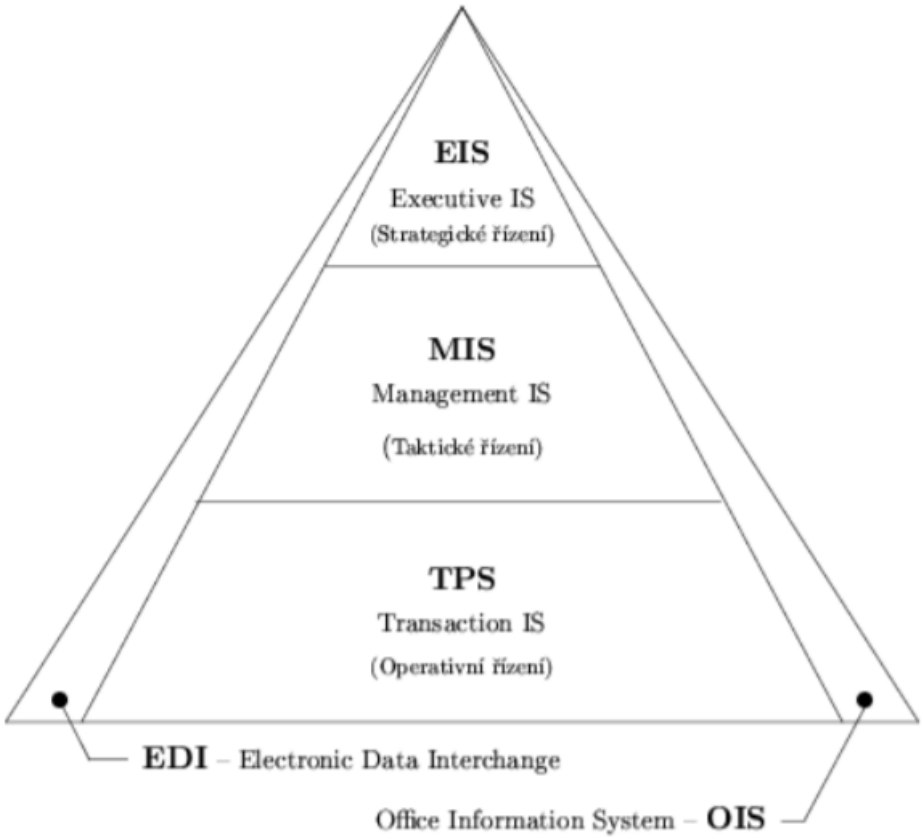
Znalostní úroveň – zahrnuje nejen klientské aplikace podnikového informačního systému (ERP, CRM, atd.), ale také prostředky osobní informatiky, jako jsou kancelářské aplikace, software určený pro týmovou práci (groupware) atd. Tyto aplikace podporují růst znalostí báze organizace a řídí především tok dokumentů. Typickými uživateli aplikací na znalostní úrovni jsou manažeři a technicko-hospodářští pracovníci na všech úrovních. [4]

Řídící úroveň – požaduje informace nutné k plnění administrativních úkolů a podpoře rozhodování zejména pak u středního a vrcholového managementu. Informační systém využívaný na řídící úrovni dává odpověď na zásadní otázku: Fungují věci tak, jak mají? Odpovědi přitom poskytuje formou tzv. reportingu, tedy generování výstupních sestav obsahujících souhrn výsledků z požadované oblasti. Podpora strukturovaného rozhodování prostřednictvím reportů probíhá nejčastěji v pravidelných intervalech. Příkladem může být reportování ekonomických výsledků z obchodní činnosti. [4]

Strategická úroveň – informační systémy pokrývající strategickou oblast bývají vrcholovému managementu nápomocny k identifikaci dlouhodobých trendů, a to jak uvnitř, tak i vně organizace. Jejich hlavní úlohou je pomoci odhalit očekávané změny a určit, zda a jak je schopen podnik na změnu reagovat. [4]

3.5 Typy IS podle úrovně řízení

Každý podnik kromě organizačních úrovní disponuje také třemi úrovněmi řízení, přičemž pro každou úroveň jsou vyžadovány jiné procesy a požadavky, které by měl informační systém zpracovávat a vyhodnocovat. K lepšímu pochopení může posloužit pyramidální struktura architektury podnikových informačních systémů, kde jednotlivé bloky (EIS – informační systémy pro podporu vrcholového vedení, MIS – manažerské informační systémy, TPS – provozní informační systémy, EDI – elektronická výměna dat a OIS – kancelářský informační systém) představují aplikace a současně reprezentují funkce aplikace, datovou základnu aplikace, aplikační software a jeho základní technologické prostředí.



Obrázek 2 Informační pyramida podpory řízení [8]

3.5.1 Provozní informační systémy (TPS)

Hlavním úkolem těchto informačních systémů je podpora hlavních činností podniku na operativní úrovni, konkrétně se jedná o provozní úroveň řízení či sledování transakcí. Historicky vychází z klasických dávkových systémů, kde pořízená data byla zpracovávána po jednotlivých dávkách. Z názvu vyplývá, že pro svůj chod využívají transakce. Kromě toho pracují jen s jednou databází. Obecně můžeme říci, že pracují s dávkovým zpracováním nebo tzv. OLTP (Online Transaction Processing), kdy u dávkového zpracování se čeká na akumulaci transakcí do jedné hromadné dávky, zatímco u OLTP se každá transakce zpracovává okamžitě bez čekání.

3.5.2 Manažerské informační systémy (MIS)

Systémy typu MIS se zabývají především řízením podniku na taktické úrovni řízení. Mezi vybrané oblasti patří obchodně logistické procesy, finančně účetní procesy či oblasti, které se vyskytují napříč celého podniku, konkrétně se může jednat např. o legislativu, řízení lidských zdrojů nebo marketing. Specifickou činností těchto systémů je evidence procesů,

zpracování ekonomických analýz či matematicko-analytické práce. Společně s TPS vypovídají o aktuálním stavu podnikových procesů.

3.5.3 Systémy pro vrcholové řízení (EIS)

Na vrcholu pyramidy se nachází tzv. systémy pro vrcholové řízení, které pracují s daty, jež jsou ve většině případů pořizována v systémech TPS a MIS. Tyto data se vyznačují vysokou agregací a strukturací. Typické funkce systémů EIS jsou:

- plánování v dlouhodobém horizontu,
- ekonomická analýza celkového hospodaření firmy,
- hodnocení podnikatelských záměrů,
- příprava inovačních akcí,
- formulace strategických projektů metodami projektového řízení,
- podpora specifikace marketingové strategie firmy,
- manažerské výkaznictví.

Důležitou součástí je DSS – systém pro podporu rozhodování, který za pomocí analýz (rozhodovací analýza a operační systémová analýza) dokáže urychlit a zpřesnit propočty klíčových rozhodnutí. Vše je přitom založeno na permanentní aktualizaci modelů, výběru a zpracování nejdůležitějších dat a na předpokladu, že uživatel rozumí podstatě metody.

3.5.4 Systémy Business Intelligence (BI)

Na serveru www.systemonline.cz se Panec [9] snaží objasnit chápaní Business intelligence jako ucelený a efektivní přístup k práci s firemními daty, který má vliv na správnost strategických rozhodnutí, a tím i na obchodní úspěch společnosti. V praxi to vypadá tak, že BI pracuje nad strukturovanou databází, ze které pomáhá zpracovat a vyhledat hodnotné informace na základě konkrétního požadavku uživatele. Ve výsledku dochází k přetváření zdrojových dat, pomocí nichž jsou následně přijímána adekvátní rozhodnutí, což vede ke strategickému posunu. Podstatnou součinnost zde hrají systémy EIS, které často vystupují v roli reprezentační vrstvy pro vybrané prostředky BI.

Softwarové produkty BI při svém nasazení poskytují svým uživatelům [7]:

Aktuální informace – o stavu dodavatelů, odběratelů, prodejů, skladů, o rozpracovanosti ve výrobě apod., bez čekání na zpracování příslušných periodických uzávěrek v transakčních systémech.

Nezávislost – protože odstraňují nutnost zjišťovat informace přes více úrovní řízení, kde může docházet k nežádoucímu „šumu“ a kde může být zpracování zbytečně zdlouhavé a případně lze čerpat současně z více datových zdrojů.

Pružnost – při dotazování na informace, které nelze specifikovat předem nebo by to bylo málo efektivní.

3.5.5 Systém pro plánování a řízení podnikových procesů (ERP)

Z informačních systémů ERP v současné době jednoznačně vyčnívá nad ostatními, respektive čerpá jednotlivé aspekty a funkcionality prakticky ze všech ostatních. Svojí velikostí již dokáže zastoupit všechny tři vrstvy úrovně řízení (EIS, MIS a TPS).

Jak uvádí Basl s Blažíčkem [7], ERP představuje softwarové řešení užívané k řízení podnikových dat a pomáhajících k plánování celého logistického řetězce od nákupu přes sklady po výdej materiálu, řízení obchodních zakázek od jejich přijetí až po expedici, včetně plánování vlastní výroby a s tím spojené finanční a nákladové účetnictví a řízení lidských zdrojů.

Z výše uvedené definice je zřejmá integrace, která stojí na samém vrcholu tohoto typu informačního systému. Integrace zásadním způsobem představuje rozdíl oproti transakčním systémům. Pod pojmem integrace si lze vybavit celou řadu funkcionalit a informací, které v sobě nesou jednotlivé moduly (finanční, lidské zdroje, logistické, marketingové apod.). Tyto moduly dokáží zastřešit vybrané agendy podniku.

Obrázek 3 Moduly ERP [10]

Sodomka a Klčová [4] vymezují ERP systém pěti základními vlastnostmi:

- automatizace a integrace hlavních podnikových procesů,
- sdílení dat, postupů a jejich standardizace přes celý podnik,
- vytváření a zpřístupňování informací v reálném čase,
- schopnost zpracovávat historická data,
- celostní přístup k prosazování ERP koncepce.

ERP systémy můžeme klasifikovat podle odborového a funkčního zaměření, přičemž rozlišujeme tyto tři kategorie:

ERP systém	Charakteristika	Výhody	Nevýhody
All-in-One	Schopnost pokrýt všechny klíčové interní podnikové procesy (personalistka, výroba, logistika, ekonomika)	Vysoká úroveň integrace, dostačující pro většinu organizací	Nižší detailní funkcionalita, nákladná customizace
Best-of-Breed	Orientace na specifické procesy nebo obory, nemusí pokrývat všechny klíčové procesy	Špičková detailní funkcionalita nebo specifická oborová řešení	Vysoké náklady, nutnost realizovat více IT projektů
Lite ERP	Odlehčená verze standardního ERP zaměřená na trh malých a středně velkých firem	Nízká cena, orientace na rychlou implementaci	Omezení ve funkcionalitě, počtu uživatelů, možnostech rozšíření atd.

Tabulka 1 Klasifikace ERP systémů podle odborového a funkčního zaměření [4]

All-in-One – dokáží pokrýt a integrovat všechny čtyři interní procesy, tj. výrobu, nákupní, prodejní, výrobní logistiku, lidské zdroje a ekonomiku. Vzhledem k poměrně jednoduchému začlenění této funkcionality do ERP řešení není organizace postavena před problém řešit další složitý integrační projekt. Dodavatel obvykle sám garantuje celé dílo včetně této subdodávky a její integrace. Volba All-in-One systému by pak pro podnik měla znamenat realizaci pouze jednoho subjektu. [4]

Best-of-Breed – do kategorie ERP řadíme také ty informační systémy, které nemusíme nutně pokrýt a integrovat všechny čtyři interní procesy. Zákazníkovi ale umí poskytnout buď detailní špičkovou funkcionalitu, nebo jsou orientované výhradně na určité obory podnikání. Tyto systémy pak v praxi bývají nasazovány buď samostatně, nebo tvoří součást podnikové ERP koncepce (procesně orientované Best-of-Breed) společně s jinými informačními systémy. [4]

Lite ERP – představují specifickou nabídku určenou pro trh malých a středně velkých podniků (SME), vyznačující se nižší cenou a nejrůznějšími omezeními. [4]

3.5.6 Systém pro řízení dodavatelského řetězce (SCM)

Systém SCM se aktivně podílí především na zajištění konkurenční výhody v oblasti dodávek, a to prostřednictvím řízení dodavatelského řetězce, díky kterému dochází ke zkracování času na zpracování a současně ke zvyšování spolehlivosti dodání produktu

zákazníkovi obecně na trh. Hlavní tok směřuje od dodavatele k zákazníkovi, přičemž mezi nimi klíčovou roli hraje výrobce, distributor a osoba či společnost, která produkt (zboží) zákazníkovi prodá, tj. prodejce. Basl a Blažíček [7] ve své knize uvádějí pět následujících komponent:

Plán – strategická část SCM nutná k řízení všech zdrojů směrem k naplnění požadavků zákazníka na výrobek nebo službu. Cílem je definování metrik pro monitorování celého řetězce tak, aby za nízké náklady dodával vysokou kvalitu a hodnotu pro zákazníka.

Nákup – výběr dodavatele materiálů, respektive služeb, potřebných pro realizaci vlastní produkce. Součástí je ocenění dodávky, dotací a platební podmínky a následné monitorování tohoto vztahu včetně jeho zlepšování.

Výroba – výroba, rozvrhování činností a operací nutných pro výrobu, testování, balení a přípravu expedice. Je to část řetězce nejvíce náročná na měření kvality, výstupů výroby a produktivity zaměstnanců.

Expedice – koordinuje příjem zakázek od zákazníka, využívá sklady a transportní možnosti k dodání produktu zákazníkovi, zajišťuje systém fakturování a placení.

Reklamace – část řetězce, která zajišťuje příjem nesprávného zboží od zákazníka a pomáhá zákazníkům, kteří mají s dodávkou produktů potíže.

3.5.7 Systém pro řízení vztahu se zákazníkem (CRM)

Podnikové informační systémy se v současnosti čím dál více zaměřují na podporu podniku v souvislosti s prodejem svých výrobků či služeb. Jedním z nejdůležitějších důvodů, proč se zavádí informační systémy, je vytváření a zlepšování vztahu se zákazníkem. Informační systémy tohoto typu oslovují jak uživatelské organizace, jimž by měly pomoci vydělat peníze, tak i dodavatele, kteří v této oblasti vidí velkou podnikatelskou příležitost.

Nové technologie 21. století, jako je internet, mobilní telefon nebo bezdrátový přenos přináší v tomto ohledu změnu ve vnímání tradiční marketingové koncepce. Na základě toho mohlo dojít k vylepšení klasických a vytvoření nových obchodních modelů. Stále častěji je kladen větší důraz na potřeby a ziskovost zákazníků, což má za následek generování poptávky pro automatizaci externích procesů (obchod, marketing, servisní služby či řízení kontaktů), a tedy pro uplatnění CRM systémů. Předpokládá, ale že [4]:

- porozumíme potřebám zákazníků,

- dokážeme je vhodně segmentovat do skupin,
- přizpůsobíme těmto skupinám produktovou nabídku a doprovodné služby,
- dokážeme rozhodnout o prioritách při automatizaci externích procesů,
- porozumíme fungování dodavatelského řetězce.

CRM systém lze správně definovat jako nástroj pro řízení CRM procesů, jehož prostřednictvím lze zabezpečit [4]:

- online přístup k informacím, který umožní pružnou reakci,
- interakce se zákazníkem a udržení kontinuity těchto interakcí,
- udržení kvality a funkcí nabízených produktů a služeb,
- identifikace ceny a možnost vytvoření cenové nabídky,
- identifikaci rozhodovacích pravomocí na straně zákazníka,
- predikci vývoje trhu a obchodní činnosti.

Pokud se podnik rozhodne k realizaci CRM koncepce, tak musí být založena na čtyřech základních pravidlech, které platí pro jakýkoli typ zamýšlené strategie. Podle Sodomky a Klčové [4] prvním pravidlem je jednotnost, neboť vůči zákazníkovi je vždy potřeba vystupovat jednotně, což v praxi znamená např. to, že zákazník nemůže zjistit odlišné konkrétní informace na sociálních sítí, než jaké jsou na oficiálních webových stránkách společnosti. Nesoulad těchto informací vyvolává v zákazníkovi pocit chaotičnost, nesrozumitelnosti a nedůvěry, což ve výsledku může společnost stát kredit, čas i peníze.

Druhé pravidlo hovoří o integrovanosti, přičemž je třeba sladit informační toky, které putují vně i dovnitř firmy všemi CRM procesy. Pokud jsou v podnikové infrastruktuře přítomny také systémy pro řízení dodavatelských řetězců (SCM) včetně pokročilého plánování a rozvrhování výroby (APS), pak můžeme celý proces „průtoku zakázky“ velmi dobře zautomatizovat. Propojení a spolupráce jednotlivých systémů zajistí to, že zákazník obdrží přehled o stavu zakázek, jejich rozpracovanosti a potřebném materiálu. V některých případech také může sledovat vývoj jejího řešení.

Třetí pravidlo se vztahuje k naplnění, konkrétně se jedná o naplnění daty, neboť data jsou nedílnou součástí všech informačních systémů. Úkolem IT oddělení je navrhnout databázovou strukturu a relaci mezi údaji. Následně již stačí zadat relevantní data do

systému, což by měli dělat obchodníci a marketingoví manažeři. Důležitým aspektem je zde kontrolovatelnost, kdy chybně uvedená data mohou vést k výsledkům, které nemusejí odpovídat realitě.

Čtvrté pravidlo se týká segmentace. Toto pravidlo říká, že každý zákazník tvoří samostatný tržní segment a vyžaduje individuální péči. Hovoříme zde s pojmy jako je homogenita (do jednoho segmentu řadíme zákazníky, kteří jsou si co nejvíce podobní svým tržním chováním) a heterogenity (definujeme jednotlivé segmenty tak, aby byly navzájem co nejvíce odlišné). Uvedené pravidlo principiálně vede k uplatňování KCRM koncepce, neboli k řízení klíčových zákazníků. KCRM se uplatní více tam, kde dominuje několik silných klientů nebo kde existuje rozsáhlý sortiment produktů.

Podobně jako u ERP systémů, i zde můžeme klasifikovat CRM systémy podle odborového a funkčního zaměření:

CRM systém	Charakteristika	Výhody	Nevýhody
All-in-One	Schopnost pokrýt všechny klíčové procesy	Vysoká úroveň integrace CRM procesů, bohatá funkcionalita	Vysoké náklady, nízká využitelnost všech funkcí
Best-of-Breed	Orientace na specifické obory nebo procesy, nemusí pokrývat všechny klíčové procesy	Špičková detailní funkcionalita nebo specifická oborová řešení	Vysoké náklady, nutnost realizovat více IT projektů
Lite CRM	Odlehčená verze standardního CRM nebo integrované CRM v rámci ERP systému	Nižší náklady	Nižší detailní funkcionalita

Tabulka 2 Klasifikace CRM systémů podle odborového a funkčního zaměření [4]

All-in-One – podobně jako ERP systém, také CRM dokáže pokrýt všechny klíčové procesy, ovšem vzhledem ke své komplexitě se vyznačuje nízkou využitelností všech funkcí.

Best-of-Breed – ačkoli tato kategorie nedokáže pokrýt všechny klíčové procesy, přesto se vyznačuje vysokými náklady.

Lite CRM – oproti předchozím kategoriím se liší v nákladovosti, kdy zavedení představuje jednoznačně nižší náklady, což se projevuje především u nižší detailní funkcionality.

3.6 Vývoj informačního systému

Vývoj informačního systému z hlediska metodiky dělíme na dva základní pilíře – tradiční a agilní. Přestože agilní metodiky jsou v posledních několika letech na vzestupu, neznamená to, že by se tradiční neboli klasické metodiky přestaly používat. Ne vždy jsou totiž agilní metodiky tím nejlepším přístupem, proto platí, že mnohdy může být vhodná kombinace obou přístupů.

3.6.1 Tradiční metodiky

Tradiční metodiky jsou velmi podrobné, formální, direktivní a společně s referenčními modely procesů, různými modely životního cyklu, posuzováním zralosti a způsobilosti procesů jsou součástí tradičních přístupů budování informačních systémů [11].

Přesně stanovují procesy, všechny požadavky a produkty, předpokládají, že tvorbu IS lze přesně definovat, popsat a opakovaně realizovat. Většina těchto metodik je založena na vodopádovém modelu životního cyklu, ve kterém jednotlivé fáze vývoje softwaru následují po sobě – více v kapitole 3.6.3 Modely životních cyklů informačních systémů.

3.6.2 Agilní metodiky

Agilní přístupy jsou opakem tradičních. Vychází z předpokladu, že proces tvorby informačního systému nelze přesně popsat, má být pružný a má nabízet rychlá řešení. Nedefinují procesy, ale popisují jen principy a praktiky a jsou tak zbaveny byrokratické zátěže. Vychází ze zkušeností získaných během vývoje, kdy je třeba projekt přizpůsobovat aktuální situaci a reagovat na změny a na požadavky zákazníka. Tyto metodiky je vhodné použít pro projekty s nejasným nebo měnícím se zadáním, pro menší týmy.

Mezi nejpoužívanější agilní metodiky patří [12]:

- eXtreme Programming (XP) – hodí se pro menší projekty a malé týmy, vyvíjející software podle zadání, které je nejasné nebo se rychle mění. Jediným exaktním, jednoznačným, změřitelným, ověřitelným a nezpochybnitelným zdrojem informací je zdrojový kód.
- Feature-Driven Development (FDD) – vývoj po malých kouscích (vlastnostech, rysech), což jsou elementární části funkcionality přinášející nějakou hodnotu uživateli. Vývoj probíhá v pěti fázích, první tři jsou sekvenční, poslední dvě pak iterativní.

- SCRUM Development Process – iterativní vývoj definující 3-8 fází (tzv. sprintu), z nich každá trvá přibližně měsíc. Nedefinuje žádné konkrétní procesy, pouze zavádí pravidelné každodenní schůzky vývojového týmu (scrum meetings), kde si jednotliví členové sdělují, které položky byly od minulé schůzky dokončeny, které nové úkoly nyní přijdou na řadu a jaké překážky stojí vývoji v cestě a musí se řešit. Každý sprint je zakončen předvedením funkční demo-aplikace zákazníkovi, který poskytne zpětnou vazbu.

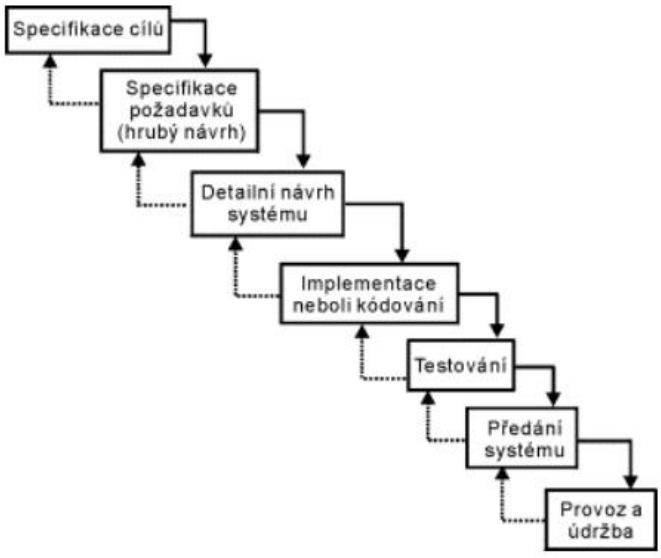
3.6.3 **Modely životních cyklů informačních systémů**

Životní cyklus systému můžeme chápat jako způsob vedení tvorby systému. V zásadě rozlišujeme hned několik typů životních cyklů IS:

3.6.3.1 Vodopádový model

Tento model patří mezi klasické modely životního cyklu používané již v 70. letech k výstavbě automatizovaných systémů řízení. Cílem jeho vzniku bylo zavést do vývoje systémů jednotný řád, umožnění řešení komplexnějších problémů díky hierarchické dekompozici a snížení množství chyb precizní kontrolou všech výstupů jednotlivých etap.

Základní charakteristikou modelu vodopád je, že při návrhu IS se provádí postupně jednotlivé etapy životního cyklu, které na sebe navazují a vzájemně se neprotínají. Etapy se provádí podle přesného plánu realizace a zpětně se k nim nevrací, dokončená etapa je vstupem etapy následující. [13]



Obrázek 4 Schéma vodopádového modelu [13]

Výhody modelu:

- tento postup je poměrně rychlý i levný pokud se nevyskytnou problémy. Je vhodné tento postup uplatnit při návrhu systému, kde je přesně známý problém a způsob jeho řešení,
- zavedení pevné struktury a kontroly do návrhu IS a ušetření lidských i finančních zdrojů.

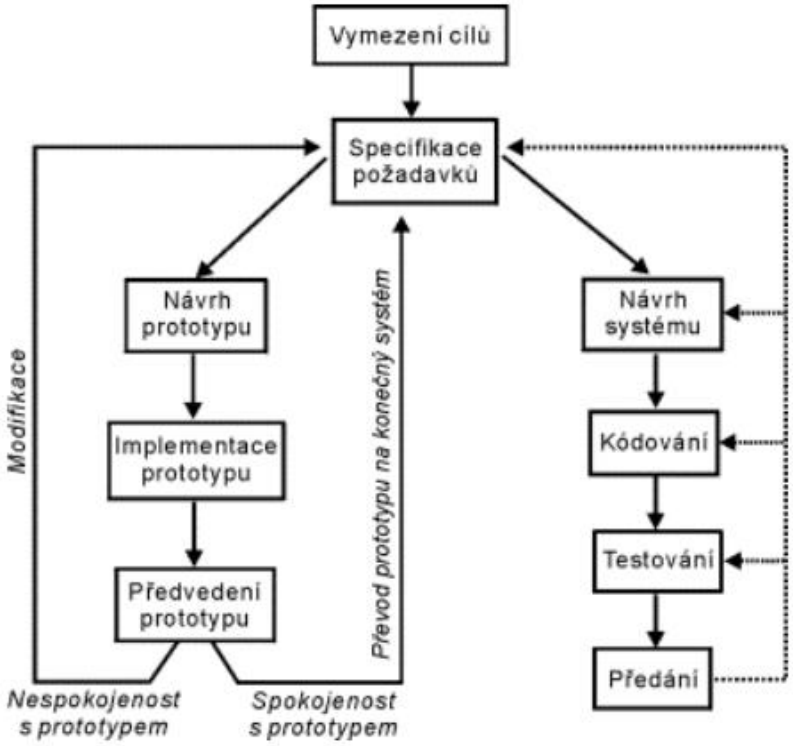
Nevýhody modelu:

- selhává u větších projektů,
- velká doba od zahájení do konce projektu,
- ztráta kontaktu s potřebami uživatelů.

3.6.3.2 Prototypový model

Tento model se začal prosazovat v 80. letech. Jeho hlavním cílem je urychlení vývoje IS využitím prototypů a seznámení zákazníka s prvními verzemi systému v co nejkratší době. Prototyp můžeme chápat jako zjednodušenou implementaci celého systému nebo jako plnou implementaci části systému. Tato implementace je provedena v co nejkratším čase a v takové funkčnosti, která prezentuje veškerá vnější rozhraní a umožňuje zákazníkovi reagovat na výsledky. Na základě připomínek zákazníků jsou upřesňovány požadavky a modifikován prototyp do té doby, dokud zákazník není spokojen. Poté následuje samotný návrh a implementace celého systému [13].

Základní charakteristikou prototypového modelu je předpoklad změn výchozích požadavků zákazníků a umožnění reakce na tyto změny, čímž se liší od modelu vodopád.



Obrázek 5 Schéma prototypového modelu [13]

Tento model nalezne své využití zejména v těchto 3 oblastech:

- simulace výkonu,
- kontrola pochopení zadání,
- návrh interface.

Výhody modelu:

- umožňuje co nejpřesněji obsáhnout požadavky budoucích uživatelů a reagovat na jejich změny,
- tato metoda je u rozsáhlých systémů poměrně náročná, proto se většinou předem určuje množství opakování prototypů a každé z nich musí být provedeno do stanoveného termínu.

3.6.3.3 Model Spirála

Tento model vytvořil B. W. Boehm v roce 1988 a je kombinací prototypového přístupu a analýzy rizik. Základem celého modelu je neustálé opakování vývojových kroků tak, že v každém dalším kroku se na již ověřenou část systému přibalují části na vyšší úrovni. Postup vývoje v jednotlivých krocích je shodný s původním modelem vodopád, přičemž každý krok se skládá z následujících částí [13]:

- specifikace cílů a určení plánu řešení,

- vyhodnocení alternativ řešení a analýza rizik s daným řešením souvisejících,
- vývoj prototypu dané úrovně a jeho předvedení a vyhodnocení,
- revize požadavků neboli validace,
- verifikace neboli ověření zda celkový výstup daného kroku je v souladu se zjištěnými požadavky.

Výhody modelu:

- model využívá ověřené kroky vývoje a analýzou rizik předchází chybám,
- umožňuje konzultovat požadavky zákazníků v jednotlivých krocích a modifikovat systém podle upřesněných požadavků,
- první verze systému je možné sledovat a hodnotit při jejich postupném vzniku.

Nevýhody modelu:

- řešení systému pomocí tohoto modelu vyžaduje neustálou spolupráci zákazníků, proto není vhodný zejména pro systémy vyvíjené na zakázku bez účasti budoucích uživatelů,
- neumožňuje přesné naplánování termínů, cen a jednotlivých výstupů, a tím i jejich plnění,
- je nutné provést bezchybnou analýzu rizik a vybrat aspekty, u nichž budeme rizika prověřovat, neboť na této analýze jsou založeny další fáze projektu. Pozdní zjištění komponent s vysokou mírou rizika může mít zásadní vliv na celý projekt,
- malá členitost modelu vyžaduje zkušené programátory, při nutnosti podrobnějšího členění je nutné zajistit precizní kontroly výstupů.

3.6.4 Jednotlivé fáze životního cyklu IS

V předchozí podkapitole jsme si představili základní členění formálních a agilních metodik pro návrh IS. Nyní se blíže podíváme, jakými fázemi vývoje systém prochází od započetí prací na jeho vzniku, až po samotný zánik systému. Tento časový úsek „života“ IS bývá často označován jako životní cyklus informačního systému, přičemž se skládá z několika fází.

Existují různé typy členění jednotlivých fází životního cyklu informačního systému, nicméně pro účely této diplomové práce budou použity zdroje od [13] [14].

3.6.4.1 Předběžná analýza

Základem celkového návrhu, vývoje i jakékoli úpravy stávajícího systému jsou požadavky uživatelů a cíle organizace. V této části se musí dané požadavky shromáždit, v hrubých rysech rozebrat a odhadnout dobu realizace a náklady. Cílem je pouze sestavit základní rámec požadavků, cílů a funkcí, nikoli je podrobněji rozebírat, to je úkolem další etapy. Výstup z této části by měl obsahovat především plány a odhady týkající se délky trvání vývoje, potřeba zdrojů, rozsahu systému, jeho klíčových požadavků a odhady návratnosti této investice.

3.6.4.2 Analýza systému

Tato část cyklu bezprostředně navazuje na předchozí část. Jejím úkolem je předchozí analýzu zpřesnit natolik, aby bylo možné odhalit všechny případné chyby ve struktuře dat i systému samotném. Chyby, které nejsou v této fázi odhaleny, se později odstraňují již jen velice těžko.

3.6.4.3 Projektová studie

Tato část je výsledkem analýzy systému. Výsledkem je dokument, který je podkladem pro obsah smlouvy s externí firmou o návrhu a realizaci IS, časový harmonogram, cena vyvíjeného projektu, konkrétní implementace systému (patří sem logický datový model a fyzický datový model), podmínky zavádění v organizaci, záruční servis a podmínky celkového předání IS.

3.6.4.4 Implementace

Tato část životního cyklu IS je vlastním programováním, kterého se účastní vybraní experti v programování a analytik nesoucí zodpovědnost za správnost řešení. Jako podklady pro jejich práci slouží veškeré informace shromážděné předchozími etapami a fyzický návrh systému.

Postup práce je následující. Na základě získaných faktů z fyzického návrhu se definují vstupy a výstupy jednotlivých operací a určí způsob jejich modifikace. Naprogramují se veškeré funkce a doladí se jejich vzájemné propojení. Dále se jednotlivé realizované funkce ověří a připraví se testovací data, která musí obsahovat maximální procento konečných reálných dat.

3.6.4.5 Testování

Informační systémy jsou testovány průběžně a nepřetržitě během celého vývoje. Přesto je však potřeba provést připravené testy na hotovém a kompletním informačním systému. Při těchto testech a zkouškách je nutné vyzkoušet veškeré možné reakce systému na data, která lze do systému zadat, odhalit případné chyby a opravit zjištěné nedostatky.

3.6.4.6 Zavádění systému

Zaváděním systému je míněna především jeho instalace, zavedení do provozu organizace, transformace původní datové základny tak, aby byla přístupná novému systému, dále poskytnutí manuálů a školení uživatelů. Při školení je nejlepším postupem nejprve školit vedoucí pracovníky a pokračovat zaměstnanci v provozu. Zavedení systému může být provedeno jedním z následujících způsobů:

- souběžná strategie,
- pilotní strategie,
- postupná strategie,
- nárazová strategie.

3.6.4.7 Rutinní provoz a údržba

Tato etapa je závěrečnou fází projektu, ve které je systém provozován a používán. Do této etapy také spadá údržba systému, tedy zajištění správného provozu, úprava parametrů aplikací nebo změny některých programů tak, aby splňovaly nové požadavky uživatelů. Mezi základní povinnosti zajištění provozu IS patří organizace prací na počítačích a v síti tak, aby byl zajištěn soulad s původním projektem a dokumentací, zajištění přístupových práv k jednotlivým aplikacím, sledování činnosti počítačů a síťových prostředků z hlediska výkonu a poruchovosti, zajištění optimálního provozu systému, zabezpečení systému a ochrana dat před neoprávněným přístupem nebo minimalizace škod vzniklých výpadkem systému, např. záložními systémy nebo archivací dat.

3.7 Jazyk UML

Existuje více různých definic jazyka UML. Jednu z nich uvádí Kanisová a Müller [15]: „Modelovací jazyk UML je souhrnem především grafických notací k vyjádření analytických a návrhových modelů. UML je jazyk, který umožňuje modelovat jednoduché i složité aplikace pomocí stejné formální syntaxe, a proto můžete výsledky své práce sdílet

s ostatními návrháři. Vybrané modely jsou pochopitelné i pro zadavatele aplikace a umožní kvalitní vyjasnění požadavků a uživatelů na vytvářený systém.“

Důležitým souslovím v definici je analytických a návrhových modelů. Dnes totiž můžeme říci, že jazyk UML se stal standardem právě v oblasti analýzy a návrhu. Pokud mluvíme o standardu, tak je důležité zmínit, že právě díky němu je usnadněna komunikace při přípravě projektů a jejich dokumentace. Z tohoto důvodu je důležité, aby se v něm programátoři orientovali. Jeho hlavní předností je usnadnění návrhu a vývoje informačního systému.

S rostoucí komplexností informačních systémů rostou nároky na samotný jazyk, který v současné době neslouží jen pro jednoho programátora, ale pro celý tým, či dokonce týmy, jež musí mezi sebou komunikovat a spolupracovat na vývoji. Primárně jazyk UML pomáhá ve fázi analýzy, kdy komunikujeme s klientem a zjišťujeme, co budeme programovat. Kromě toho je nesmírně užitečný ve fázi designu, kdy naopak řešíme otázku, jak to naprogramujeme.

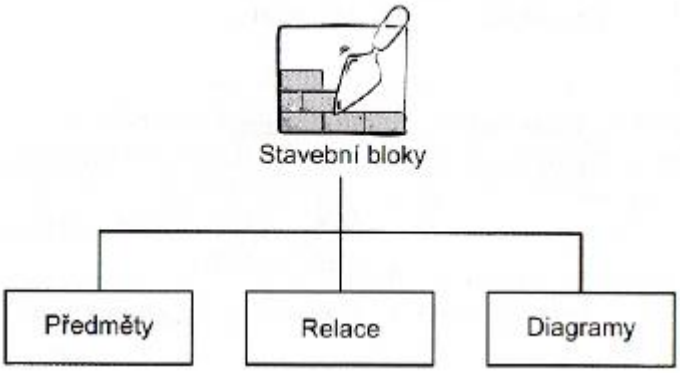
3.7.1 Struktura jazyka UML

Arlow a Neustadt [16] uvádějí, že pro lepší pochopení jazyka UML jako jazyka vizuálního, podíváme-li se na jeho strukturu. Struktura jazyka UML obsahuje tyto části:

- Stavební bloky – jsou to základní prvky modelu, relace a diagramy,
- Společné mechanismy – obecné způsoby, jimiž v jazyku UML lze dosáhnout specifických cílů,
- Architektura – pohled v jazyku UML na architekturu navrhovaného systému.

Podle definice Pavla Tišnovského [17] je celý jazyk UML založený na třech elementech, které ale nejsou z uživatelského hlediska reprezentovány v textové podobě, ale grafickými značkami v plošném (tj. dvourozměrném) grafu. Tyto tři základní elementy jazyka UML se dle své funkce nazývají:

- Předměty – představují samotné elementy modelu,
- Vztahy (relace) – jsou pojítkem mezi předměty, tj. určují, jak spolu dva nebo více předmětů významově souvisí,
- Diagramy – představují pohledy na modely UML, neboť obsahují kolekci předmětů, které vizualizují to, co systém bude dělat, a to, jak to bude dělat.



Obrázek 6 Stavební bloky jazyka UML [16]

3.7.1.1 Předměty

Předměty (rovněž „věci“ nebo abstrakce) jsou elementy zpracovávaného modelu, jež jsou následně členěny do několika navzájem rozdílných podkategorií [17].

- **Strukturní abstrakce** – jedná se o podstatná jména modelu UML, například programové třídy, aplikační či objektové rozhraní, případy užití, komponenty či uzly. V diagramu UML jsou strukturní abstrakce zobrazeny jako různé převážně plošné tvary, které jsou však vždy uzavřené. Například se jedná o obdélníky, elipsy, kružnice či jednoduché zdánlivě trojrozměrné tvary, například krychle či kvádry. Každý smysluplný UML diagram by měl obsahovat alespoň dvě strukturní abstrakce, při jedné abstrakci totiž modelování ztrácí svůj hlavní smysl – popis vztahů mezi jednotlivými objekty.
- **Chování** – představují slovesa modelu UML, prezentují interakce, tj. vzájemné komunikace mezi jednotlivými objekty. Pomocí chování lze také modelovat stavový stroj, u nějž se stavy specifikují pomocí přechodů, událostí a aktivit. Chování se v UML diagramu většinou vyznačuje pomocí různým způsobem konstruovaných a různě zakončených šipek či propojovacích čar.
- **Seskupení** – graficky seskupuje části diagramu na nižší úrovni. Většinou se jedná o takzvané balíčky, jež mají tvar stylizované kancelářské složky s popisem umístěným v levé horní části obdélníku (zobrazení seskupení se však může v různých prostředích odlišovat). Seskupení nejsou v současné době v některých dále popisovaných aplikacích použitelná (tj. nelze vytvářet hierarchické diagramy), což je pro tvorbu rozsáhlejších grafů velké omezení.

- **Poznámky –** specifikují vlastnosti a chování dalších elementů UML diagramu. Graficky jsou poznámky na prakticky všech typech UML diagramů většinou vyvedeny žlutě vyplněným obdélníkem s ohnutým růžkem.

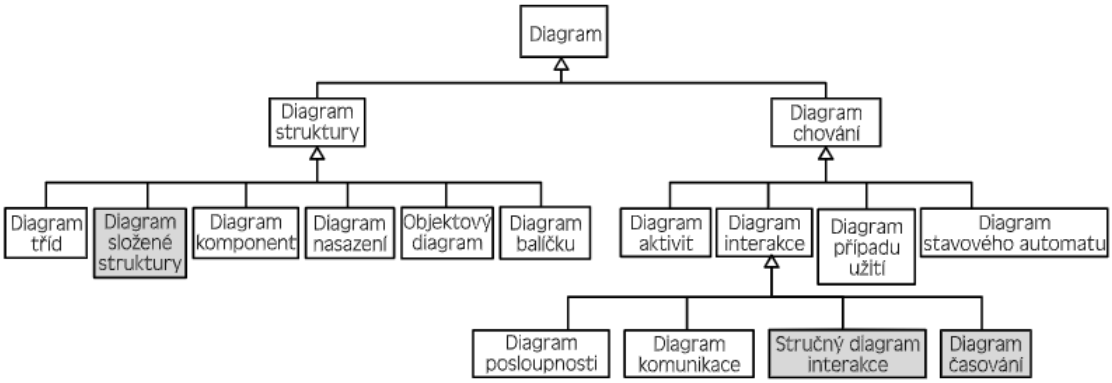
3.7.1.2 Relace

Vzhledem k tomu, že v grafech je zapotřebí předměty různým způsobem navzájem propojovat, jsou v jazyku UML specifikovány i relace, tj. vztahy mezi různými předměty. V UML jsou rozeznávány následující podtypy relací [17].

- **Asociace** – pomocí asociací se modeluje obecná souvislost předmětů, která je však v diagramu UML přesným způsobem definovaná. Speciální variantou asociace jsou takzvané kompozice a agregace, které jsou často používány v objektově orientovaných jazycích a návrzích databází.
- **Závislost** – použije se, pokud změna v jednom předmětu způsobí změnu v předmětu jiném, nebo mu známým způsobem poskytne požadovanou informaci.
- **Generalizace** – pomocí generalizace se modeluje stav, kdy je jeden předmět specializací jiného předmětu. Tato relace je velmi často používána v objektově orientovaných jazycích, implementuje se většinou pomocí dědičnosti (inheritance).
- **Realizace** – jedná se o druh vztahu, ve kterém jeden předmět představuje dohodu, za jejíž splnění je odpovědný jiný předmět. V objektově orientovaných jazycích se realizace vytváří pomocí rozhraní (interface) – samozřejmě za předpokladu, že daný OOP jazyk rozhraní podporuje.

3.7.1.3 Diagramy

Diagramy zachycují různé aspekty modelovaného systému, který nemusí být obecně vyjádřen pouze jedním UML diagramem, protože je možné, například pomocí balíčků, sdružovat více diagramů do jednoho hierarchicky organizovaného celku. V zásadě rozlišujeme dva modely – statický (diagram struktury) a dynamický (diagram chování), od kterých jsou odvozeny další diagramy – viz obr. 7 níže.



Obrázek 7 Členění diagramů v UML [16]

Diagram tříd

Pro správné pochopení je nutné si správně definovat pojem třída. Pro návrh objektových systémů se předpokládá, že je třeba navrhnout model tříd objektů, který v podstatě nezobrazuje jednotlivé objekty, ale šablonu (předpis) pro vytvoření objektů – to je třída objektů. Třída objektů je definována svými atributy a metodami. Při návrhu třídy neuvažujeme o konkrétním naplnění atributů, pouze definujeme jejich název a typ. Při vzniku instance objektu se atributům přiřadí skutečné hodnoty.

Z výše uvedené definice lze vyvodit vztah mezi třídou a objektem. Pokusme se tedy předložit definici pojmu objekt: „Objekt je seskupením dat a funkcionality, které jsou spolu spojeny za účelem plnění soudržné množiny zodpovědností. Objekt má svou identitu, vlastnosti, chování a zodpovědnost. Vlastnosti jsou v podstatě atributy objektu a chování objektu je realizováno jeho metodami. Každý objekt má jedinečnou zodpovědnost (dovednost).“ [15]

V průběhu analytických prací hledáme objekty v problémové oblasti, jsou to abstrakce, koncepty nebo věci. Objekty mají jasné hranice a nacházejí se v problémové oblasti. Objekty mají velice často přímý odraz v reálném světě a uvedený odraz lze nalézt zkoumáním problémové oblasti. Některé objekty, jako účetní doklady, či zboží, mohou být konkrétní věci, jiné mohou být naopak abstraktní, jako typy dokladů, typy zboží. Klíčem je, aby každý objekt měl jasně definované zodpovědnosti, které mu umožní poskytovat vzájemně související služby.

Objekt poskytuje služby prostřednictvím operací. Každá operace je jednotkou práce vykonanou objektem. Objekty spolu komunikují předáváním zpráv. To eliminuje mnoho

datových duplicit a zároveň zjišťuje, že změny struktury dat jsou zapouzdřeny uvnitř objektů a nerozšiřující svůj dopad do dalších částí systému. [15]

Diagram tříd je nejčastěji používaným digramem při modelování objektově orientovaných systémů a představuje statický pohled na strukturu modelovaného systému. Znázorňuje množinu tříd, rozhraní a jejich vzájemnou spolupráci a vztahy. [18]

V první úrovni namodelujeme pouze základní logickou strukturu systému, která odpovídá požadavkům na systém. Tento model se nazývá doménový a obsahuje základní třídy, atributy a vztahy mezi třídami. Druhým v pořadí je návrhový model a rozšiřuje doménový model o datové typy atributů, operace a doplňující třídy (například výčtové třídy). Dále upřesňuje asociace mezi třídami. Poslední model je implementační a zaměřuje se na detailní popis struktury. Je primárně určen pro vývojáře systému. Rozšiřuje návrhový model o speciální operace, tzv. *gettery* a *settery*, které slouží k nastavení a získávání hodnot atributů. Dále přidává manažerské třídy a neposlední řadě shlukuje třídy do balíků. [19]

Prvky diagramu:

- **Dědičnost, generalizace** – generalizace je klíčový koncept objektově orientované analýzy a návrhu. Pokud nacházíme v analýze tento typ vztahu, tak jej implementujeme v objektovém prostředí jako dědění. Princip dědění umožňuje objektovým třídám sdílet jejich charakteristiky včetně hierarchie dědění a uchovat jejich rozdíly. Generalizace se především používá z důvodů opakovaného použití, aby se nemusely opakovat společné prvky. V podstatě je generalizace vztahem mezi obecnější objektovou třídou a více upřesněnou další objektovou třídou, která následuje v hierarchii dědění na nižší úrovni. Dědičnost může být i ve více než jedné úrovni. [15]
- **Asociace** – znázorňuje vztahy mezi jednou či více třídami, které jsou abstrakcí množiny spojení mezi instancemi (objekty) těchto tříd. U asociací předpokládáme, že jsou v podstatě obousměrné, pokud nejsou explicitně specifikovány jako jednosměrné. Jméno asociace implikuje směr a mělo by být v aktivní (slovesné) formě. Obvykle se čte zleva doprava a shora dolů. Asociaci dále můžeme rozdělit na:
    - **Agregace** – agregace je vazbou typu, kdy jeden objekt je součástí jiného objektu, zatímco u asociace hovoříme o rovnoprávnějším vztahu. U agregace

nadřízený objekt využívá dovedností podřízeného objektu a měl by nést zodpovědnost za jeho vznik a zánik.

  - **Kompozice** – speciálním typem agregace je kompozice, kdy víme, že podřízený objekt nemůže existovat samostatně bez nadřízeného objektu.

- **Násobnost** – násobnost nebo multiplicita určuje, kolik výskytů jedné třídy může mít vztah k jednomu výskytu přiřazené třídy. Multiplicita je omezení rozsahu souborů objektů, kardinalita je současný počet objektů v souboru. [20]

Diagram případů užití

Diagram případů užití je jeden z klíčových diagramů pro modelování dynamických vlastností systému. *Use case* diagramy, jsou klíčové k modelování vlastností systému, subsystému nebo tříd. Jednotlivé diagramy zobrazují množinu případů užití a aktérů a jejich vzájemné propojení. Jsou důležité pro vizualizaci, specifikaci a dokumentaci systému. [18]

Prvky diagramu [16]:

- **Účastníci** – jsou to role, přidělené vnějším entitám (osoby, jiné systémy), které přímo komunikují se systémem. Aktérem je velmi často i čas.
- **Případy užití** – činnosti, které systém vykonává jménem jednotlivých aktérů nebo v jejich prospěch. Případy užití vždy iniciuje aktér a jsou vždy psány z jeho pohledu.
- **Relace** – vztahy mezi aktéry a případy užití. Relace může být obousměrná nebo jedno směrná
- **Hranice systému** – ohraničení zobrazené kolem případů užití, jež je vyznačením hranic modelovaného systému. Aktéři se nacházejí mimo hranice systému.

Sekvenční diagramy

Digramy interakce popisují vzájemnou spolupráci skupiny objektů. UML definuje několik interakčních diagramů, avšak nejčastěji používaným je sekvenční diagram. Typicky tento diagram zachycuje zasílání zpráv mezi klasifikátory v rámci případu užití. [21]

Sekvenční diagramy znázorňují klasifikátory na horní hraně diagramu. Na horní liště vlevo je umístěn aktér, který je zodpovědný za spuštění případu užití. Z klasifikátorů vede svislá čára reprezentující život objektu v průběhu daného případu užití. V terminologii UML je tako čára nazývávána jako čára života (LifeLine). Každá zpráva mezi objekty je znázorněna šipkou mezi čárami života. Zprávy jsou požadavky objektu o vyvolání operace

druhého objektu. Objekty mohou zasílat zprávy i samy sobě (tzv. self-call). Pořadí (sekvence) zasílání zpráv je reprezentováno osou Y, čas plyne od shora dolů. [15]

Prvky diagramu [16]:

- **Čára života** – reprezentuje, jakým způsoben se instance určitého klasifikátoru účastní interakce. V hlavičce čáry života je uveden klasifikátor.
- **Zpráva** – zastupuje druh komunikace mezi dvěma čárami života.
- **Kombinované fragmenty** – oblast uvnitř sekvenčního diagramu s odlišným chováním.
    - **Alt –** podmínka definuje, zda dojde k provedení operace.
    - **Loop** – cyklus, smyčka.

Diagram aktivit

Diagram aktivit jsou „objektově orientovanými diagramy toků“. Díky nim lze modelovat proces jako kolekci aktivit a přechodů mezi nimi. Diagramy aktivit jsou ve skutečnosti obdobou stavových diagramů, v nichž stavy reprezentují vykonávání aktivit a přechody jsou vyvolány ukončením aktivity. Diagram aktivit lze připojit k libovolnému modelovanému elementu. Diagramy aktivit jsou obvykle připojeny k [16]:

- případům užití,
- třídám,
- rozhraním,
- komponentám,
- uzlům,
- spolupracím,
- operacím a metodám.

Diagram aktivit lze s velkým úspěchem použít rovněž k modelování obchodních (podnikatelských) procesů a pracovních postupů. Přestože jsou diagramy určeny především k řazení aktivit, může být skutečný zdrojový kód operace nejlepším a nejpregnantnějším vyjádřením. Vždy je třeba rozhodovat podle podstaty problému. Diagram aktivit je možné použit při modelování business procesů, ale v dnešní době je v této oblasti spíše nahrazen

technikami BPMN (Business Process Model and Notation) nebo notací Eriksson – Penker, která je rozšířením UML.

Stavový diagram

Stavový diagram poskytuje pohled na dynamické chování objektu v průběhu času pomocí modelování životního cyklu dané třídy, jinými slovy modelují chování objektu napříč případy užití. Jednotlivé objekty reagují na události v systému a podle toho mění své stavy. Stav je množina hodnot atributů pro danou třídu [18]. Stavové diagramy jsou velice podobné diagramům aktivit, liší se však sémantikou i účelem. Diagramy aktivit vycházejí z Petriho sítí a jsou určeny k modelování procesů.

Prvky diagramu [19]:

- **Stav** – reprezentace konkrétní situace, která nastala během životního cyklu objektu. Tato situace je dále charakterizována tak, že během ní objekt vyhovuje nějaké podmínce, realizuje konkrétní operaci nebo třeba jen čeká na příchod události, která proběhne během zpracovávání stavu. Stav může dále obsahovat akce a aktivity. Za akci považujeme nepřerušitelný, rychle probíhající proces, zatímco aktivita je přerušitelný, jistou dobu trvající proces.
- **Přechod** – posun z jednoho stavu do jiného vyvolaného určitou událostí.
- **Události –** specifikace něčeho významného, co nastane v určitém čase a prostoru. Lze ji znázornit externě na přechodech nebo interně uvnitř stavů. Kanisová a Müller [15] ve své knize dělí události na čtyři základní podtypy:
- **Událost volání** – patří mezi nejjednodušší typy událostí. Odpovídá metodě dané třídy, je vlastně požadavkem na její spuštění. Událost volání tak ve svém důsledku spouští sérii akcí.
- **Signální událost** – reprezentuje příjem asynchronně předávané zprávy mezi objekty. Zpravidla se signály modelují jako samostatné třídy se stereotypem <<*signal*>> a veškeré předávané informace mají uloženy ve svých atributech.
- **Změnová událost** – se skládá z klíčového slova *when* (ekvivalentu když), podmínky a akce. Události tohoto typu jsou aktivovány tehdy, pokud je splněna logická podmínka.

- **Časová událost** – bývají uvozeny klíčovými výrazy *when* nebo *after*. Událost může být vygenerována po určité době, toto lze indikovat klíčovým slovem *after*.

Dále existují dvě speciální události *entry* a *exit*. Jakákoliv akce, která je napojena na událost *entry*, je automaticky spuštěna, kdykoliv se do daného stavu dostaneme přechodem. Akce asociované s událostí *exit* jsou analogicky provedeny, kdykoliv je daný stav opuštěn přechodem.

3.8 Požadavky

Arlow a Neustadt [16] ve své knize definují požadavky jako základ všech systémů, nebo by alespoň měl být, neboť jsou v podstatě vyjádřením toho, co by měl systém dělat. Požadavky by měly být jediným vyjádřením, *co* by měl systém dělat, nikoli toho, *jak* by to měl dělat. V tomto oba autoři vidí nesmírně důležitý rozdíl. Můžeme určit, *co* by měl systém dělat a jaké chování by měl poskytovat, aniž bychom cokoli říkali o způsobu, *jak* bude dané funkce dosaženo.

Požadavek lze definovat jako specifikaci toho, co by mělo být implementováno. V podstatě rozlišujeme dva typy požadavků:

- funkční požadavky, jež určují, jaké chování bude systém nabízet,
- nefunkční požadavky, které specifikují vlastnosti nebo omezující podmínky daného systému.

3.8.1 Zdroje požadavků

Nežli se pustíme do samotného vytváření diagramů a celkového návrhu informačního systému, je zapotřebí se nejprve věnovat požadavkům na daný systém. Zpravidla míváme sami menší či větší představu o tom, co by budoucí systém měl poskytovat, jak by měl fungovat, přesto drtivá většina požadavků pochází od budoucích uživatelů.

Proces získávání požadavků od budoucích uživatelů je poměrně náročnou disciplínou. Můžeme se buďto setkat s uživateli, kteří mají ve svých myšlenkách poměrně jasně vykreslenou představu, jaké služby od systému očekávají, přičemž své představy jsou navíc schopni dostat do realizovatelné podoby. Druhou skupinu uživatelů tvoří ti, kteří přenechají veškerou aktivitu na tvůrcích systémů a požadují dodání takového systému, který uspokojí všechny jejich nároky, aniž by spolupracovali na jejich konkretizaci. Ve výsledku

se mnohdy stává, že uživatelé se systémem nejsou spokojeni, přičemž neskrývají výhrady, což má za následek např. časovou prodlevu termínu dodání.

Jako zdroje požadavků můžeme identifikovat zejména [15]:

- legislativu,
- požadavky zákazníků (písemné, získané konzultacemi, …),
- existující systémy uživatelů,
- pracovní procesy uživatelů,
- vlastní know-how pro danou problémovou oblast,
- prostředí zákazníka,
- hardwarové a softwarové vybavení.

3.8.2 **Techniky sběru požadavků**

Pro účely této práce definujeme dvě základní techniky pro sběr požadavků: interview a JAD (Joint application design). Obě techniky ukazují problémy k řešení v novém systému, umožňují řešitelům pochopit prostředí a zmenšují možnost opomenutí požadavků. Přesto mezi nimi existují zásadní rozdíly, které budou rozebrány v následujících dvou podkapitolách:

3.8.2.1 Interview

Představuje systematickou snahu shromáždit informace od jedné osoby, přičemž posláním je dozvědět se kompetentní odpovědi. Chybně provedené interview může způsobit negativní účinek celého projektu. Kvalitní interview se skládá z následujících kroků:

- Příprava – je založena na pochopení organizace a fyzické prohlídce.
- Plánování a načasování – příprava otázek, kdy lze odeslat předběžný dotazník a určit, kdo bude tázán a v jakém pořadí – začínáme nejvyšším vedením.
- Zahájení a závěr – interview většinou zahajuje analytik, který se představí a sdělí účel interview. Vhodnější je vyhnout se uzavřeným otázkám či dlouhým poznámkám. Na závěr stručně shrnout diskutované okruhy, vysvětlit některé věci, poděkovat a požádat o další interview.
- Provedení – největší důraz je zde kladen na tzv. techniku aktivního naslouchání, která je charakteristická dotazy s otevřeným koncem, použitím vhodných slov a frází,

znamení akceptace formou neverbální komunikace, přeformulování odpovědí či efektivní užití mlčení.

Snažíme se klást otevřené otázky, neboť u tohoto typu otázek se mnohdy zjistí informace, které nebyly záměrem. Ne vždy mohou být vhodné, důvodem je potřeba explicitního potvrzení odpovědi.

3.8.2.2 JAD (Joint application design)

Oproti interview získáváme informace od skupiny (nikoliv od jednotlivce). To může vytvářet pocit součinnosti, která může vést až k větší zodpovědnosti. Debatujeme převážně s kvalifikovanými uživateli a IS pracovníky. Je zde menší riziko chyb v požadavcích, jelikož se bavíme s odborníky, proto je menší pravděpodobnost chybné interpretace. Rozlišujeme šest typů účastníků:

- Uživatelé – klíčoví účastníci, u kterých se setkáváme s pocitem součinnosti.
- Analytici – naslouchají uživatelům a formulují jejich požadavky. Jsou z projekčního týmu.
- Pozorovatelé – poskytují technickou pomoc a radu.
- Zapisovatelé – dokumentují průběh jednání. Je u nich přípustný zvukový záznam, jelikož cokoli kdo řekne, tak je všeobecně známé stanovisko.
- Moderátor – zajišťuje hladký průběh semináře. Vše závisí na jeho schopnostech, podstatou je svobodné vyjádření a konsensus. Zároveň pro něj platí, že by neměl být podřízeným nikoho ze skupiny, nesmí mít rozhodující hlas a musí usilovat o shodu.
- Výkonný sponzor – zahajuje a uzavírá seminář, vyjadřuje podporu top managamentu.

3.9 Programovací jazyk Java

Kapitola se zabývá objektově orientovaným programovacím jazykem Java. Úvodní část kapitoly bude věnována základním informacím o jazyku Java, historii, původnímu účelu a momentálnímu využití. V dalších kapitolách bude blíže vyjádřen vztah k OOP (objektově orientované programování) a samotnému fungování jazyka.

3.9.1 Základní informace o jazyku Java

Javu lze charakterizovat jako silně typový, generický, objektově orientovaný jazyk založený na třídách. Představený byl 23. května 1995 společností Sun Microsystems a za jejího autora je považován James Arthur Gossling. Verze 1.0 vyšla 23. ledna 1996.

Původním účelem Javy byl vývoj programů na přenosném zařízení, avšak v dnešní době je využíván zejména na webové aplikace založené na architektuře klient-server. Jazyk se stal populární zejména proto, že byl schopen zkompilovat běh programu na kterékoliv platformě, která podporuje Javu, bez nutnosti rekompilace zdrojového kódu. V dnešní době je vývoj a distribuce Javy pod křídly společnosti Oracle.

3.9.2 **Historie jazyka Java**

Java vznikla jako moderní programovací jazyk, který měl za cíl odstranit problémy do té doby nejrozšířenějšího objektově orientovaného jazyka – C++. Práce na vývoji začala v roce 1991. Při vývoji bylo stanoveno vícero cílů. Jazyk byl od začátku stavěný jako objektově orientovaný, přičemž v tehdejší době si podobné jazyky získaly poměrně velkou oblibu, protože umožňovali moderní přístup ke tvorbě programů. Oproti C++, Java neobsahovala nízkoúrovňovou práci s pamětí. Koncept paměti byl navržen velmi jednoduše. Nové objekty se vytvářejí pomocí klíčového slova *new* a o místo se stará automatická správa paměti (garbage collector). Napříč tomu si však jazyk zachoval syntax, která byla velmi podobná C++. To umožňovalo přechod na nový programovací jazyk bez výrazných problémů. [22]

Dalším z cílů bylo vytvořit jazyk takový, aby zkompilovaný kód běžel nezávisle na platformě. Jazyk C++ byl schopný běžet na kterékoliv platformě, avšak pro každou novou platformu bylo potřebné kód programu překompilovat. Java tento problém odstranila. Java a její kód je zkompilovaný do platformy nezávislého byte kódu (bytecode). Navíc Java specifikuje velikost svých základních datových typů a správu aritmetických operátorů. Díky tomu nemůže dojít k nekompatibilitě datových typů na různých platformách. [22]

Posledním z cílů bylo vytvořit Javu tak, aby byla více vláknová a dynamická. Java tak nabízí synchronizované primitivní datové typy, třídu *Thread* na manuální práci s vlákny a vícero systémovými knihami, které umožňují paralelní běh programu ve vícero vláknech. Zajímavostí je, že každý program napsaný v Jave je vícero vláknový, protože automatická správa paměti běží v samostatném vlákně s nízkou prioritou. Dynamičnost Javy se projevuje zejména ve fázi vázání. Třídy jsou navázané pouze v případě, kdy jsou potřebné, přičemž moduly mohou být navázané i při běhu programu. [22]

3.9.3 Hlavní atributy jazyka Java

Herbert Schildt [23] ve své knize zmiňuje návrhový tým jazyka Java, který shrnul klíčová hlediska do následujícího seznamu atributů:

- **Jednoduchost** – jazyk Java obsahuje stručnou a soudržnou sadu funkcí, díky níž se jej lze snadno naučit a používat,
- **Bezpečnost** – jazyk Java umožňuje vytvářet bezpečné internetové aplikace,
- **Přenositelnost** – programy Java lze spouštět v libovolném prostředí, kde existuje systém runtime tohoto jazyka,
- **Objektová orientace** – v jazyce Java se projevuje moderní objektově orientovaná filozofie programování,
- **Robustnost** – jazyk Java pomáhá programovat bez chyb, protože má striktní typovou kontrolu a provádí kontroly za běhu,
- **Vícero vláken** – jazyk Java poskytuje integrovanou podporu programování s více vlákny,
- **Neutralita vzhledem k architektuře** – jazyk Java není svázán s určitou architekturou hardwaru nebo operačního systému,
- **Interpretovaný kód** – díky využití bajtového kódu je kód jazyka Java přenositelný mezi platformami,
- **Vysoký výkon** – bajtový kód Java je optimalizován na rychlost spouštění,
- **Distribuované prostředí** – jazyk Java je optimalizován na rychlost spouštění,
- **Dynamičnost** – programy Java obsahují mnoho informací o typech runtime, které slouží ke kontrolám a zpřístupnění objektů za běhu.

3.9.4 Objektově orientované programování (OOP)

Jazyk Java je založen na objektově orientovaném programování. Objektově orientovanou metodologii nelze od jazyka Java oddělit a všechny programy v tomto jazyce jsou alespoň do určité míry objektové. OOP je silný nástroj k řešení programátorských úkolů. Od doby prvních počítačů se metodologie programování značně změnila, zejména kvůli rostoucí složitosti programů. [23]

Obecně lze říci, že v každé fázi vývoje programování vznikaly techniky a nástroje, které programátorům umožňovaly zvládat stále větší složitost. Každý nový přístup přebíral nejlepší prvky předchozích metod a otevíral nové možnosti. Před příchodem OOP se mnoho projektů dostávalo do bodu, kde strukturovaný přístup přestával fungovat (nebo tento bod dokonce překračovalo). Objektově orientované metody byly vyvinuty proto, aby programátorům pomáhaly takové bariéry překonat. [23]

Na základě principů objektově orientovaného programování mají všechny objektové jazyky, včetně jazyka Java, tři společné vlastnosti: zapouzdření, polymorfizmus a dědičnost.

3.9.4.1 Zapouzdření

Zapouzdření je programátorský mechanizmus, který svazuje kód a data, s nimiž manipuluje, a obojí zabezpečuje před vnějšími vlivy a neoprávněným přístupem. V objektově orientovaném jazyce lze kód s daty propojit takovým způsobem, že vzniká samostatná *černá skříňka*. Tato skříňka obsahuje všechna potřebná data a kód. Takovým sloučením kódu a dat vzniká objekt. Jinak řečeno: objekt je mechanizmus, který zajišťuje zapouzdření. [23]

Kód, data, případně kód i data, objektu mohou být vzhledem k němu *soukromé* nebo *veřejné*. O soukromém kódu či datech vědí a mohou k nim přistupovat pouze jiné části objektu. Pro úseky programu, které existují vně programu, tedy soukromé kódy ani data nejsou přístupné. V případě veřejného kódu či dat k nim mohou jiné části programu získat přístup, i když jsou příslušné prvky definovány uvnitř objektu. Veřejné části objektu obvykle poskytují kontrolované rozhraní pro soukromé prvky objektu. Základní jednotkou zapouzdření jazyka Java je *třída* (class). Třída definuje formu objektu. Určuje data i kód, který bude s těmito daty pracovat. Jazyk Java pomocí specifikace třídy konstruuje *objekty*. Objekty jsou instancemi třídy. Třída tedy v zásadě odpovídá sadě plánů, které popisují, jakým způsobem vytvořit objekt. [23]

3.9.4.2 Polymorfizmus

Polymorfizmus označuje vlastnost, která jednomu rozhraní poskytuje přístup k obecné třídě akcí. Konkrétní akce závisí na přesné povaze situace. Příkladem může být zásobník (což je seznam, ze kterého se první prvek odebírá jako poslední). Určitý program může vyžadovat tři různé druhy zásobníků. Jeden zásobník slouží pro celočíselné hodnoty, jiný pro desetinná čísla a další pro znaky. Všechny zásobníky jsou v tomto případě

implementovány pomocí stejného algoritmu, ačkoli ukládaná data se liší. V jazyce, který není objektově orientovaný, bychom museli vytvořit tři různé sady rutin zásobníku a každou z nich označit jinými názvy. Díky polymorfizmu však v jazyku Java stačí vytvořit jednu obecnou sadu rutin zásobníku, která se uplatní ve všech třech situacích. [23]

3.9.4.3 Dědičnost

Dědičnost je proces, kdy může jeden objekt získat vlastnosti jiného objektu. Tento princip je důležitý, protože je základem hierarchické klasifikace. Většinu informací o světě dokážeme zvládat díky tomu, že je setřídíme do hierarchie (tj. orientované struktury). Bez hierarchií by každý objekt musel explicitně definovat všechny své vlastnosti. Díky dědičnosti stačí, když objekt definuje pouze vlastnosti, které jsou v rámci jeho třídy jedinečné. Své obecné atributy může zdědit od nadřazené třídy. Právě mechanismus dědičnosti tedy umožňuje, aby jeden objekt představoval určitou instanci obecnějšího případu. [23]

3.9.5 **Základní pojmy ve vztahu k UML**

Úkolem této kapitoly je definovat základní pojmy, které se vyskytují jednak v jazyce UML, tak v jazyce Java.

3.9.5.1 Konstruktor

Konstruktor je metoda volaná při vytváření instance třídy, typicky se využívá pro inicializaci objektu. Název konstruktoru je stejný jako název třídy, konstruktor nemá návratový typ a nelze ho volat přímo jako jiné metody. Deklarace konstruktoru není povinná. Při vytváření instance třídy, která nemá definovaný žádný konstruktor, se automaticky vytvoří implicitně prázdný konstruktor (bez parametrů). Konstruktor je automaticky vyvolán při vytvoření instance pomocí klíčového slova *new*. [24]

Konstruktor je možné definovat také s parametry, to je výhodné pro snadnější inicializaci objektů (není nutné nastavovat každý atribut zvlášť). Parametrický konstruktor navíc de facto vynutí zadání parametrů a vždy provede inicializaci. Je tedy vhodný pro vynucení zadání hodnot, na které se nesmí zapomenout. Parametrický konstruktor je také možné využít v souvislosti se zapouzdřením. Pokud má objekt atributy, které nelze v průběhu života objektu měnit, pak je výhodné vytvořit k nim metody pouze pro čtení a jejich počáteční nastavení zajistit prostřednictvím konstruktoru s parametry. [24]

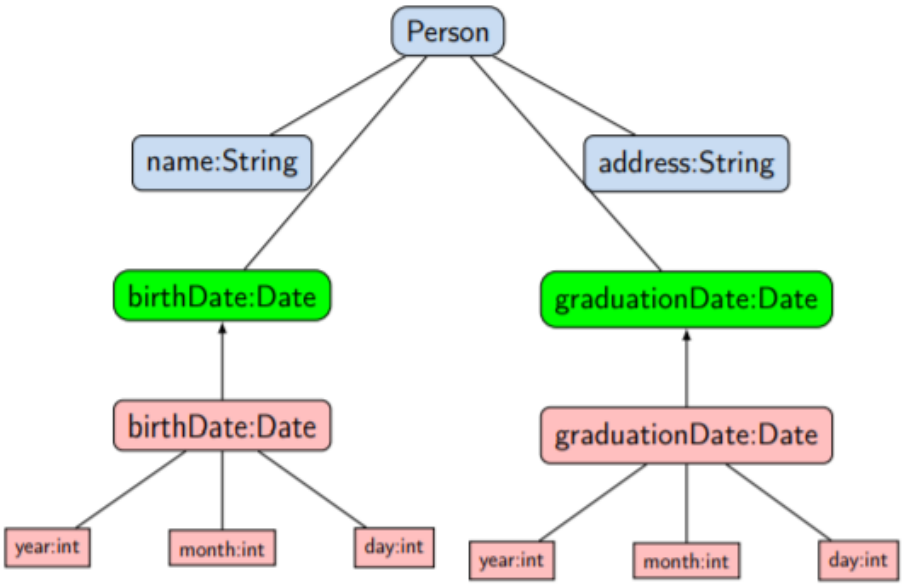
3.9.5.2 Rozhraní

Rozhraní (interface) je množina metod, která může být implementována třídou. Interface pouze popisuje metody, jejich vlastní implementace však neobsahuje. V Javě na rozdíl od jiných programovacích jazyků (například C++) nemůže třída dědit od více tříd najednou (neexistuje vícenásobná dědičnost). Každá třída však může implementovat libovolný počet rozhraní, do jisté míry tedy rozhraní vícenásobnou dědičnost nahrazují. Implementace rozhraní není na hierarchii tříd nijak vázána a nevzniká z ní vztah dědičnosti. [25]

Rozhraní může kromě definic metod obsahovat konstanty. Ty se pak chovají stejně, jako by se jednalo o konstanty třídy, která toto rozhraní implementovala. Podobně jako lze pomocí dědičnosti rozšiřovat třídy, lze rozšiřovat i rozhraní. Pokud vytvoříte rozhraní, které dědí od jiného rozhraní, automaticky tak přebírá všechny jeho metody a konstanty. Implementovat rozhraní znamená implementovat všechny metody, které toto rozhraní definuje. Pokud třída implementuje nějaké rozhraní, je tím zaručeno, že obsahuje definici všech, které rozhraní definuje. Implementace rozhraní se uvádí v záhlaví definice třídy klíčovým slovem *implements*. [25]

3.9.5.3 Kompozice

Obsahuje-li deklarace třídy členské proměnné objektového typu, pak se jedná o kompozici objektů. Kompozice vytváří hierarchii objektů. Příklad kompozice je znázorněn na obrázku č. 8.



Obrázek 8 Příklad kompozice [26]

3.9.5.4 Agregace

Agregace představuje volnou vazbu mezi celkem a součástí, kdy jeden objekt (celek) využívá služby dalších objektů (součástí). Jedná se o více specializované verze vztahu přidružení.

4 Praktická část

Praktická část je rozdělena do jednotlivých podkapitol, které na sebe chronologicky navazují. Nejprve dojde k zodpovězení klasické otázky – pro koho samotný systém děláme? Následovat bude sepsání a charakterizování konkrétních požadavků. Definice jednotlivých požadavků bude přetvořena do jazyka UML, konkrétně do diagramu tříd, který bude reprezentovat statickou strukturu systému, znázorňovat datové struktury, operace u objektů a základní souvislosti mezi objekty. Na závěr dojde k základní transformaci diagramu tříd do programátorského jazyka Java.

4.1 Pro koho je informační systém určen

Celý návrh informačního systému by se měl týkat jednoho nejmenovaného soukromého knihkupectví v pražských Dejvicích. Knihkupectví disponuje jednou pobočkou, kde si zákazníci mohou prohlížet, vybírat a následně případně také zakoupit vybrané tituly. Ačkoli knihkupectví má již více než stovky spokojených zákazníků, navíc se může těšit ze stále nových a nových zákazníků, čím dál více pociťuje absenci ve formě internetového e-shopu. Tomuto faktu nahrává také současná pandemická situace, kvůli které může mít knihkupectví otevřené pouze výdejní okénko. Je tedy na místě navrhnout informační systém tak, aby dokázal propojit kamennou prodejnu s internetovým e-shopem, díky čemuž by zákazníci mohli využívat služeb a nabídek knihkupectví z pohodlí domova.

4.2 Sběr požadavků formou interview

Pro sběr požadavků, které jsou nezbytné pro objektový model systému, byla vybrána technika interview neboli strukturovaný rozhovor. Vzhledem k tomu, že se jedná o případovou studii, tak interview bude vedeno se dvěma odborníky, kteří budou prezentováni jako expert 1 a expert 2. Jednotlivá interview budou s každým odborníkem uskutečněna samostatně tak, aby nedošlo k ovlivnění či přímé změně názoru.

Otázka č. 1: *Jak se budou zákazníci moci přihlásit ke svému zákaznickému účtu?*
Expert 1: Možnost přihlášení by měla být viditelná na úvodní stránce knihkupectví ve formě tlačítka *„přihlásit se“*. Po kliknutí na tlačítko by se mělo otevřít nové okno, kde vyskočí dvě prázdná pole k vyplnění – e-mail a heslo. Jakmile zákazník vyplní potřebné údaje, klikne na tlačítko *„přihlásit“* a opět se vrátí na úvodní stránku, ovšem nyní už jako přihlášený zákazník.

Otázka č. 2: *Jakým způsobem budou chráněny hesla a další citlivé údaje zákazníků?*
Expert 2: Ochrana hesel a citlivých údajů je pro nás jedna z priorit, alespoň co se zabezpečení týče. Jsme si vědomi toho, že šifrování nestačí, proto chceme údaje chránit pomocí hashovací funkce, která ze vstupu vytvoří takový výstup, z kterého již nelze odvodit původní řetězec. Další příjemná vlastnost hashovacích funkcí je, že generují výstupy vždy se stejnou délkou a malá změna na vstupu vždy kompletně změní celý výstup.

Otázka č. 3: *Jakým způsobem bude zákazníkovi umožněn přístup do klubového účtu?*
Expert 1: Zákazník by měl mít přístup ke klubovému účtu skrze hlavní stránku, přičemž ve svém klubovém účtu by se měl dozvědět základní informace o sobě a také o klubové kartičce, kterou zákazníci využívají při osobním nákupu přímo v knihkupectví.

Otázka č. 4: *Které jazyky bude IS podporovat?*
Expert 2: Uživatelské rozhraní IS bude primárně podporovat češtinu. Výjimku budou tvořit nepřeložené knihy, u kterých bude název a jejich popis v angličtině.

Otázka č. 5: *Na základě čeho budou charakterizovány vybrané kategorie knih a kde tyto kategorie budou zákazníkovi přístupné?*
Expert 1: Členění knih do vybraných kategorií bude vycházet ze zkušeností z osobního prodeje, tj. fyzické prodejny, kde jsou knihy rozčleněny pro jednodušší vyhledání do jednotlivých kategorií tak, aby to pro zákazníky bylo co nejvíce pohodlné. Kategorie budou přístupné opět na hlavní stránce knihkupectví.

Otázka č. 6: Jak bude zajištěn přehled o prodaných knihách?
Expert 2: Přehled o prodaných knihách bude zajištěn pomocí výkazu prodeje, který bude vždy v určitých intervalech generován.

Otázka č. 7: *Co bude následovat po kliknutí na vybranou knihu zákazníkem?*
Expert 1: Zákazník by měl být přesměrován na novou stránku, která bude patřit výhradně vybrané knize. Zde by měly být základní údaje o knize, jako např. název, cena, jméno autora, základní popis nebo nakladatelství. Samozřejmě, že podobných údajů chceme o knize evidovat mnohem více. Více praktických informací o knize může zákazníka v konečném

důsledku utvrdit v nákupu. Na druhou stranu nechceme mít u žádné knihy nerelevantní údaje.

Otázka č. 8: *Která data budete chtít ukládat a kam?*
Expert 2: V první řadě chceme evidovat co nejvíce relevantních dat o zákazníkovi, následně o jednotlivých objednávkách a nakonec o samotných knihách. Data by měla být ukládána do databáze MS SQL.

Otázka č. 9: *Jak bude moci zákazník přidat knihu do košíku a jakým způsobem se bude moci následně vrátit k nákupu či pokračovat k samotné objednávce?*
Expert 1: U každé knihy bude tlačítko *„přidat do košíku“*, poté, co zákazník klikne, se zobrazí nové okno, ve kterém uvidí základní přehled knih, jež jsou v košíku. Ve spodní hraně okna bude mít následně na výběr ze dvou možností: *„pokračovat v nákupu“* a *„přejít k objednávce“*.

Otázka č. 10: *Jak plánujete zálohovat data, případně v jakých intervalech?*
Expert 2: Data plánujeme zálohovat na fyzický server specializovaný pro práci s daty. Co se týče frekvence či konkrétních intervalů zálohování, tak záleží na typu dat, ovšem ty základní bychom chtěli zálohovat na denní bázi.

Otázka č. 11: *Co bude součástí vytvoření objednávky?*
Expert 1: Základním stavebním kamenem vytvoření objednávky je výběr dopravy. Zákazník si zde bude muset vybrat, zda si objednávku bude přát doručit na pobočku, nebo na uvedenou adresu. Jakmile zvolí možnost doručení na konkrétní adresu, systém by měl automaticky rozpoznat, zda se adresa nachází v Praze, nebo mimo Prahu. Na základě výsledku by měl systém automaticky vypočítat cenu za dopravu. Doručení na pobočku by mělo stát 30 Kč, doručení po Praze 80 Kč a doručení mimo Prahu 130 Kč. Po vyplnění všech ostatních údajů by měl zákazník přistoupit k zaplacení objednávky.

Otázka č. 12: *Počítáte do budoucna s rozvojem IS, pokud ano, s jakým?*
Expert2: Ano, do budoucna určitě počítáme s novými moduly, které by rozšířily nově vzniklý IS. Příkladem může být dodavatelský modul.

Otázka č. 13: *Jakým způsobem se bude kontrolovat průběh a stav objednávky?*

Expert 1: Poté, co zákazník vyplní potřebné údaje objednávky, tak dojde k automatické validaci, zda jsou všechny tyto údaje správně vyplněny. Systém by měl také umožnit identifikovat stav, ve kterém se momentálně objednávka nachází, příkladem může být stav nová, doručovaná, zrušená, apod.

Otázka č. 14: *Co vše budou muset servery zajistit?*
Expert 2: Především udržet IS v provozu, ale také zajistit vnitropodnikové procesy, zálohu dat, bezpečnost, sdílení souborů či stabilitu.

Otázka č. 15: *Jaké možnosti bude mít zákazník při platbě objednávky?*
Expert 1: Základní možností, jak zaplatit svoji objednávku rychle a bezpečně, bude skrze platební kartu. Jelikož ale knihkupectví chce podporovat Bitcoin a jiné kryptoměny, jakožto moderní technologie, tak jako alternativa k platební kartě by měla existovat možnost zaplacení objednávky skrze kryptoměny Bitcoin, Ethereum a Litecoin.

4.3 Požadavky na informační systém

Na základě výše uvedených otázek a odpovědí pro experty bude přistoupeno k definici samotných požadavků na systém. Z teoretické části víme, že základní dělení požadavků je na funkční a nefunkční.

4.3.1 Funkční požadavky

Definují základní úkoly nebo akce, které by měl být schopen informační systém vykonávat. Na základě interview definujeme tyto funkční požadavky:

Složitost na škále 1-10 (1 = nejlehčí, 10 = nejtěžší).
Priorita na škále 1-10 (1 = nejméně důležitá, 10 = nejvíce důležitá).

Číslo	Název požadavku	Složitost	Priorita	Návaznost
1	Přihlášení zákazníka	5	10	
2	Vstup do klubového účtu	2	5	1
3	Výběr knihy podle kategorie	3	7	
4	Zobrazení obsahu u jednotlivých knih	4	10	
5	Přidání knihy do košíku	3	10	
6	Vytvoření objednávky	8	10	5
7	Kontrola objednávky	5	8	6
8	Platba objednávky	8	10	6

Tabulka 3 Seznam funkčních požadavků, zdroj: vlastní

- **Přihlášení zákazníka** – zákazník se přihlašuje do knihkupectví skrze nové okno, ve kterém pro úspěšné přihlášení musí vyplnit kolonku email a heslo. Autorizovaný email zákazník zadává při prvotní registraci. Heslo zadává aktuální, tj. takové, jaké si naposledy v systému uložil. Heslo by mělo obsahovat minimálně šest znaků, jedno velké písmeno a číslo. Po úspěšném přihlášení by se zákazník měl dostat na úvodní stránku knihkupectví.
- **Vstup do klubového účtu** – zákazník by měl mít kdykoli z hlavní stránky přístup ke svému klubovému účtu, kde by se měl být schopen dozvědět informace následující typu: jaký je jeho bodový stav klubového konta, historii jeho objednávek, jaké knihy přečetl, jaké knihy plánuje přečíst, seznam e-knih, základní údaje a nastavení.
- **Výběr knihy podle kategorie** – na úvodní stránce knihkupectví by v nabídce měla být přístupna záložka kategorie. Jakmile by zákazník najel na tuto záložku, automaticky by se mu rozvinula nabídka s kategoriemi, jako např. odborná literatura, světová beletrie, rozvoj osobnosti atd.
- **Zobrazení obsahu u jednotlivých knih** – pokud uživatel se chce dozvědět více informací o konkrétní knize, po kliknutí na obálku vybrané knihy, by se mu měla načíst nová stránka, kde budou základní informace o knize, jako např. název, cena, typ vazby, jazyk, počet stránek, datum vydání, nakladatel, ISBN či dostupnost. Kromě toho by se měl jednoduše dostat k informacím o autorovi, který knihu napsal.
- **Přidání knihy do košíku** – u každé knihy by měl mít zákazník možnost kliknout na tlačítko *„přidat do košíku“*. Po kliknutí na tlačítko by se mu mělo otevřít nové okno, kde by měl vidět názvy přidaných knih, jejich počet, cenu za kus a dostupnost. Dále by měl mít na výběr, zda bude pokračovat v nákupu, nebo se přesune k samotné objednávce.
- **Vytvoření objednávky** – v novém okně by zákazník měl nejprve vybrat jednu z možností dopravy. Buďto by zvolil možnost odeslání knihy přímo na pobočku, nebo by napsal adresu, na kterou si přeje knihu poslat. Následně by přistoupil ke kontrole objednávky v podobě zkontrolování všech potřebných údajů. Jakmile by všechny údaje byly správně vyplněné, zákazník by mohl tlačítkem *„zaplatit objednávku“* přistoupit k zaplacení objednávky.
- **Kontrola objednávky** – kontrola spočívá jednak při samotném vytváření objednávky, přičemž je zapotřebí zkontrolovat, zda zákazník náležitě vyplnil

všechny potřebné kolonky, které jsou označené červenou hvězdičkou, ale také v dalších fázích, jako je připravenost objednávky k expedici, doručení či zrušení.

- **Platba objednávky** – jedná se o poslední krok objednávky. Zákazník má zde na výběr ze dvou základních možností platby – kartou nebo kryptoměnou. Při platbě kartou bude následně přesměrován na 3D Secure technologii, která slouží k většímu zabezpečení internetových plateb. Při platbě kryptoměnou vybere kryptoměnu, kterou chce zaplatit, následně zvolí svoji adresu (burzu) a postupuje dále dle dalších pokynů.

4.3.2 Nefunkční požadavky

Definují požadavky na architekturu informačního systému, uživatelské rozhraní a prostředí pro běh informačního systému. Na základě interview definujeme tyto funkční požadavky:

Složitost na škále 1-10 (1 = nejlehčí, 10 = nejtěžší).
Priorita na škále 1-10 (1 = nejméně důležitá, 10 = nejvíce důležitá).

Číslo	Název požadavku	Složitost	Priorita
1	Česká lokalizace	3	5
2	Generování prodaných knih	7	9
3	Ukládání dat	5	8
4	Zálohování dat	6	8
5	Rozšiřitelnost	6	8
6	Správa hesel	7	10
7	Servery	9	10

Tabulka 4 Seznam nefunkčních požadavků, zdroj: vlastní

- **Česká lokalizace** – veškeré uživatelské prostředí informačního systému bude lokalizováno do českého jazyka. Výjimku mohou tvořit zahraniční nepřeložené tituly, včetně jejich popisů.
- **Generování prodaných knih** – pro knihkupectví je zejména z pohledu účetního důležité mít přehled o všech prodaných knihách. Na základě získaných hodnot lze mimo jiné také vyvodit užitečné informace, které mohou vést například k rozšíření sortimentu knih ve vybraném odvětví.
- **Ukládání dat** – práce s daty je pro jakýkoli informační systém klíčová. Z tohoto důvodu je zapotřebí ukládat data do databáze. V tomto případě bude využita databáze MS SQL.

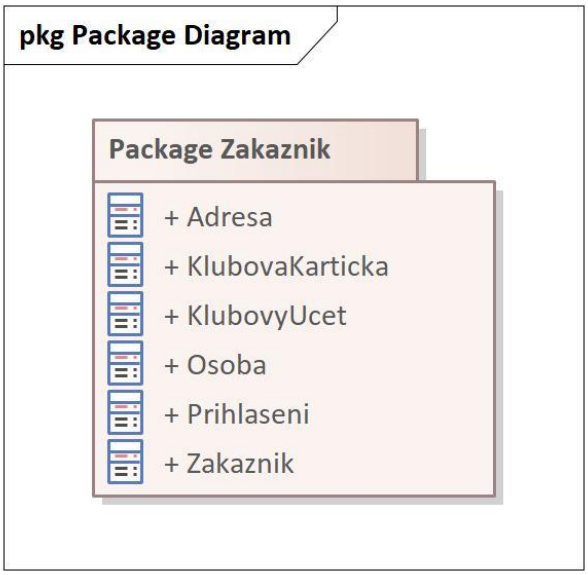
- **Zálohování dat** – o zálohování dat se starají tzv. zálohovací procedury, které musí být nastaveny tak, aby bylo zajištěno zálohování veškerých důležitých dat vytvořených uživateli v intervalech, které odpovídají rozsahu dat a jejich důležitosti. Dále musí být zálohovány systémové informace, u kterých musí být datována frekvence, rozsah a způsob zálohování.
- **Rozšiřitelnost** – rozšiřitelností je zde myšlena možnost rozšířit informační systém o nové moduly tak, aby nové moduly byly kompatibilní s těmi současnými a zároveň nenarušily jejich dosavadní chod.
- **Správa hesel** – úkolem informačního systému by mělo být uchovávat hesla uživatelů, která jsou nezbytná pro jejich přihlášení. Hesla by neměla být uchovávána čitelně, jelikož si je kdokoli může přečíst, ale ani šifrovaně, jelikož se mohou kdykoli zpátky dešifrovat. Ideálním způsobem, jak ochránit hesla, tak je za použití hashovací funkce, kdy ve skriptovacím jazyce PHP se můžeme setkat např. s funkcí md5() nebo sha1(). K heslům by zároveň neměl mít nikdo přístup.
- **Servery** – jelikož komunikace probíhá formou klient-server, je zapotřebí vystavit funkční server, který bude zajišťovat jednak samotný běh, ale i další činnosti informačního systému.

4.4 Logický model informačního systému

Tato kapitola se týká fáze analýzy v životním cyklu projektu informačního systému nazývaném návrh datového modelu. V návrhovém modelu tříd jsou definovány atributy a operace se základními typy. Analytické asociace jsou zde rozpracovány do agregací a kompozic. Kromě reprezentace statického uspořádání základních typů objektů a jejich vztahů, je nezbytné přesně vědět, jak budou informace ukládány a organizovány. Vytváření konkrétních tříd bude vycházet ze základních požadavků na systém. Nástrojem pro tvorbu bude Enterprise Architect ve verzi 15.2. od společnosti Sparx Systems, který kromě systémové analýzy a návrhu, dokáže pokrýt celý životní cyklus vývoje systému.

4.4.1 Návrhový model informačního systému – zákazník

Při návrhu jednotlivých tříd u zákazníka vycházíme z požadavků, které jasně říkají, že o zákazníkovi chceme evidovat jen podstatné informace, které nám v budoucnu mohou být užitečné, např. při statistických analýzách apod. Níže nalezneme seznam jednotlivých tříd, které ve výsledku tvoří ucelený diagram tříd.



Obrázek 9 Package zakaznik, zdroj: vlastní

Třída zákazník – jedná se o základní třídu, která obsahuje následující atributy:

- id zákazníka – jednoznačně identifikuje zákazníka.
- oblíbený žánr – eviduje žánr, který má zákazník nejraději, přičemž na základě toho je mu následně vybírána doporučená literatura.
- klubová kartička – zde vlastní či nevlastní klubovou kartičku (váže se ke klubovému členství).
- je členem klubu – určuje, zda je, či není zákazník členem klubu.
- preferované doručení – zákazník má na výběr výhradně ze dvou možností, kam mu bude kniha doručena – na pobočku nebo domů.
- datum registrace – kdy zákazník provedl registraci.

Třída osoba – samozřejmě, že o zákazníkovi lze evidovat mnohem více informací, proto vznikla třída osoba, ze které třída zákazník dědí následující atributy:

- jméno,
- příjmení,
- email,
- datum narození,
- místo bydliště,
- telefonní číslo.

Klubová kartička – v atributech třídy zákazník byl definován atribut klubová kartička. Jelikož o klubové kartičce můžeme zaznamenat vícero informací, vznikla proto třída

klubová kartička, která je ke třídě zákazník ve vztahu kompozice, tj. kdyby zanikl zákazník, zanikne také klubová kartička daného zákazníka. Třída má následující atributy:

- id kartičky – jednoznačně identifikuje klubovou kartičku.
- jméno zákazníka – uchovává jméno zákazníka, kterému patří.
- datum vydání – kdy byla kartička vydána zákazníkovi.
- platnost – dokdy kartička platí.

Adresa – další třída, která obdobně jako klubová kartička, je ve vztahu kompozice, tak je třída adresa, která v sobě uchovává základní atributy, které se týkají aktuální adresy zákazníka. Tato třída má následující atributy:

- ulice,
- město,
- číslo popisné,
- poštovní směrovací číslo.

Klubový účet – tato třída je asociována s třídou zákazník, přičemž obsahuje následující atributy:

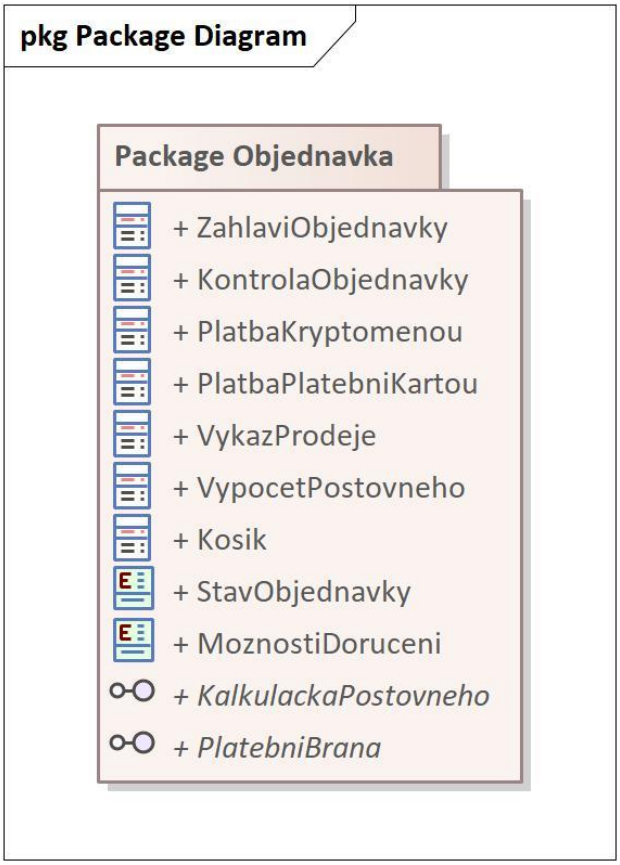
- stav konta – říká, kolik bodů má zákazník na svém kontě.
- objednávky – ukazuje aktuální, vyřízené a zrušené objednávky.
- moje knihovna – zde se zákazníkovi zobrazí jednotlivé tituly, které již přečetl.
- e-knihovna – zde se zákazníkovi zobrazí jednotlivé online tituly, které již přečetl.
- nastavení – zde si zákazník může přenastavit základní nastavení, jako je např. změna hesla apod.
- datum vytvoření – eviduje datum založení klubového účtu.

Přihlášení – úzce souvisí s předchozí třídou, tj. klubový účet. Zahrnuje následující atributy, které jsou stěžejní pro přihlášení:

- email,
- heslo.

4.4.2 Návrhový model informačního systému – objednávka

Již z požadavků lze odvodit, že objednávka se z pohledu návrhu informačního systému nachází mezi zákazníkem a knihou. Objednávka je také z pohledu návrhu nejvíce komplexní částí, která skýtá hned několik tříd a vazeb.



Obrázek 10 Package objednávka, zdroj: vlastní

Záhlaví objednávky – tuto třídu můžeme nazvat středobodem celého návrhu, jelikož ze všech tříd má nejvíce vazeb. Tato třída pracuje s následujícími atributy:

- id objednávky – jednoznačná identifikace konkrétní objednávky.
- datum objednávky,
- místo doručení – zákazník volí mezi doručením na pobočku a svoji adresou.
- poštovné – je vypočítáno na základě toho, zda objednávka bude doručována na pražskou pobočku, na adresu, která se nachází na území Prahy, nebo se adresa nachází mimo Prahu.
- slevový poukaz – knihkupectví nabízí během roku několik slevových poukazů, které mohou zákazníci kdykoli uplatnit.
- studentská sleva – vztahuje se primárně pro studenty s platným ISIC průkazem.

Třída obsahuje také tyto operace:

- záhlaví objednávky – volá argument zákazník, přičemž slouží pro vypsání informací o zákazníkovi.
- obsah košíku – pracuje s obsahem košíku. Předmětem jsou opět knih, které zákazník přidal do košíku.

- věkové omezení – vrací *boolean*, tj. zda zákazník nakupuje knihu, která je věkově omezená, či není.

Košík – typem spojení kompozice je tato třída propojena s třídou záhlaví objednávky, která v podstatě plní záhlaví objednávky. Tato třída obsahuje základní identifikátor toho, co a kolik si zákazník chce objednat. U třídy definujeme tyto atributy:

- id knihy,
- počet – určuje množství knih v košíku.
- cena celkem – určuje cenový součet všech vybraných knih.

Kontrola objednávky – typem spojení <<*create*>> je k záhlaví objednávce připojena právě tato třída, která obsahuje výhradně pouze následující operace:

- průběh objednávky – jako argument přijímá třídu záhlaví objednávky.
- vytvoření objednávky – jako argument přijímá třídu kniha. Návratovým typem je zde záhlaví objednávky.
- správnost objednávky – kontroluje, zda v objednávce jsou všechny údaje validní. Vrací *boolean*.

Stav objednávky – třídu kontrola objednávky lze rozšířit ještě o speciální typ třídy <<enumeration>>, která obsahuje tyto hodnoty:

- nová,
- připravená,
- doručovaná,
- doručená,
- zrušená.

Platební brána – jedná se o typ <<*interface*>>, jenž obsahuje operaci proces platby, která v sobě nese údaje o zákazníkovi a částku k zaplacení. Platební brána je dále realizována třídami platba kryptoměnou a platba platební kartou.

Platba kryptoměnou – charakterizuje možnost provést zaplacení objednávky pomocí kryptoměn. Platba kryptoměnou má následující atributy:

- URL – představuje novou stránku, kde dochází k zaplacení pomocí kryptoměn.
- apiKey – slouží k ověření uživatelů a identifikaci účtu, ke kterému automatizovaný program přistupuje.

Kromě atributů obsahuje třída také operaci s názvem platba kryptoměnou, která pomocí apiKey umožňuje realizovat platbu.

Platba platební kartou – Druhá jmenovaná třída obsahuje operaci autentizace a tyto atributy:

- cílová adresa – slouží pro přesměrování na platební bránu konkrétní banky, kde následně dojde k vyzvání zákazníka o zadání potřebných údajů z jeho platební karty.
- port – váže se k cílové adrese, slouží v počítačových sítích při komunikaci pomocí protokolů TCP a UDP k rozlišení aplikace v rámci počítače.
- 3D Secure – souvisí se zabezpečením platby. Funguje jako dodatečný způsob identifikace držitele karty.

Třída obsahuje také operaci autentizace, která představuje schválení platby ze strany banky držitele karty, tj. platícího zákazníka.

Kalkulačka poštovného – jelikož vyjadřuje pouze chování, proto je typem stereotypu <<*interface*>>, přičemž obsahuje operaci výpočet poštovného, která má za úkol spočítat výši poštovného.

Výpočet poštovného – charakterizuje základní výši cen, které jsou nabízeny za jednotlivé možnosti doručení knih. Výše ceny je definována v následujících atributech:

- doručování po Praze – cena 80 Kč.
- doručování mimo Prahu – 130 Kč.
- na pobočku – 30 Kč.

Pro výpočet poštovného je stěžejní vzdálenost, tj. jak daleko zákazník bydlí od pobočky respektive, zdali bydlí v Praze nebo mimo Prahu. K tomu slouží operace výpočet vzdálenosti, která pro svůj výpočet pracuje s adresou zákazníka, který je zde jako argument.

Možnosti doručení – jedná se o typ stereotypu <<*enumeration*>> a třída výpočet poštovného jej využívá pro charakterizování možností doručení. Obsahuje tyto hodnoty:

- po Praze,
- mimo Prahu,
- na pobočku.

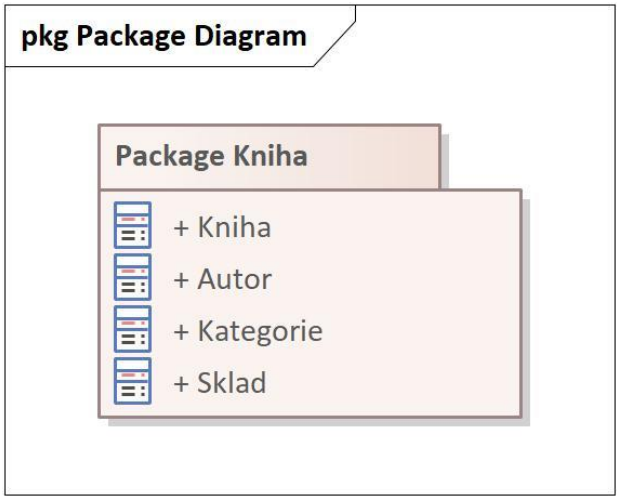
Výkaz prodeje – tato třída má za úkol generovat prodej knih, což v dalších fázích analýzy může být např. přetvořeno do konkrétních reportů. Je spojena typem agregace s třídou záhlaví objednávky. Třída disponuje těmito operacemi:

- výkaz prodeje – bere v potaz seznam všech objednávek, které k určitému datu byly uskutečněny.
- generování prodejů – dokáže vygenerovat všechny objednávky, které k určitému datu byly realizovány.

- zobrazení prodejů – slouží pro klasické zobrazení prodejů, návratový typ je *void*, tudíž nevrací žádnou hodnotu.

4.4.3 Návrhový model informačního systému – kniha

Kniha je základním stavebním kamenem knihkupectví. Z požadavků vyplývá, že největší důraz při návrhu by měl být kladen na obsah jednotlivých knih a členění do vybraných kategorií.



Obrázek 11 Package kniha, zdroj: vlastní

Kniha – jedná se o základní třídu, která obsahuje následující atributy:

- id knihy – jednoznačně identifikuje knihu.
- název,
- popis – přináší základní popis o tom, o čem kniha je, jakých témat se dotýká apod.
- cena,
- bestseller – určuje, zda se jedná o tzv. bestseller, tj. knihu, která je mezi zákazníky obecně velmi populární a prodávaná.
- typ vazby – určuje, zda se jedná o pevnou, brožovanou či jinou vazbu.
- skladem,
- jazyk,
- počet stran,
- datum vydání,
- nakladatel,
- ISBN.

Třída kniha obsahuje kromě výše uvedených atributů také jednu operaci:

- je věkově omezená – říká, zda kniha má věkové omezení (18 a více let), nebo je dostupná všem věkovým kategoriím.

Autor – knihu obvykle napíše jeden nebo více autorů, z toho důvodu definujeme třídu autor, která je spojena s třídou kniha formou asociace a má následující atributy:

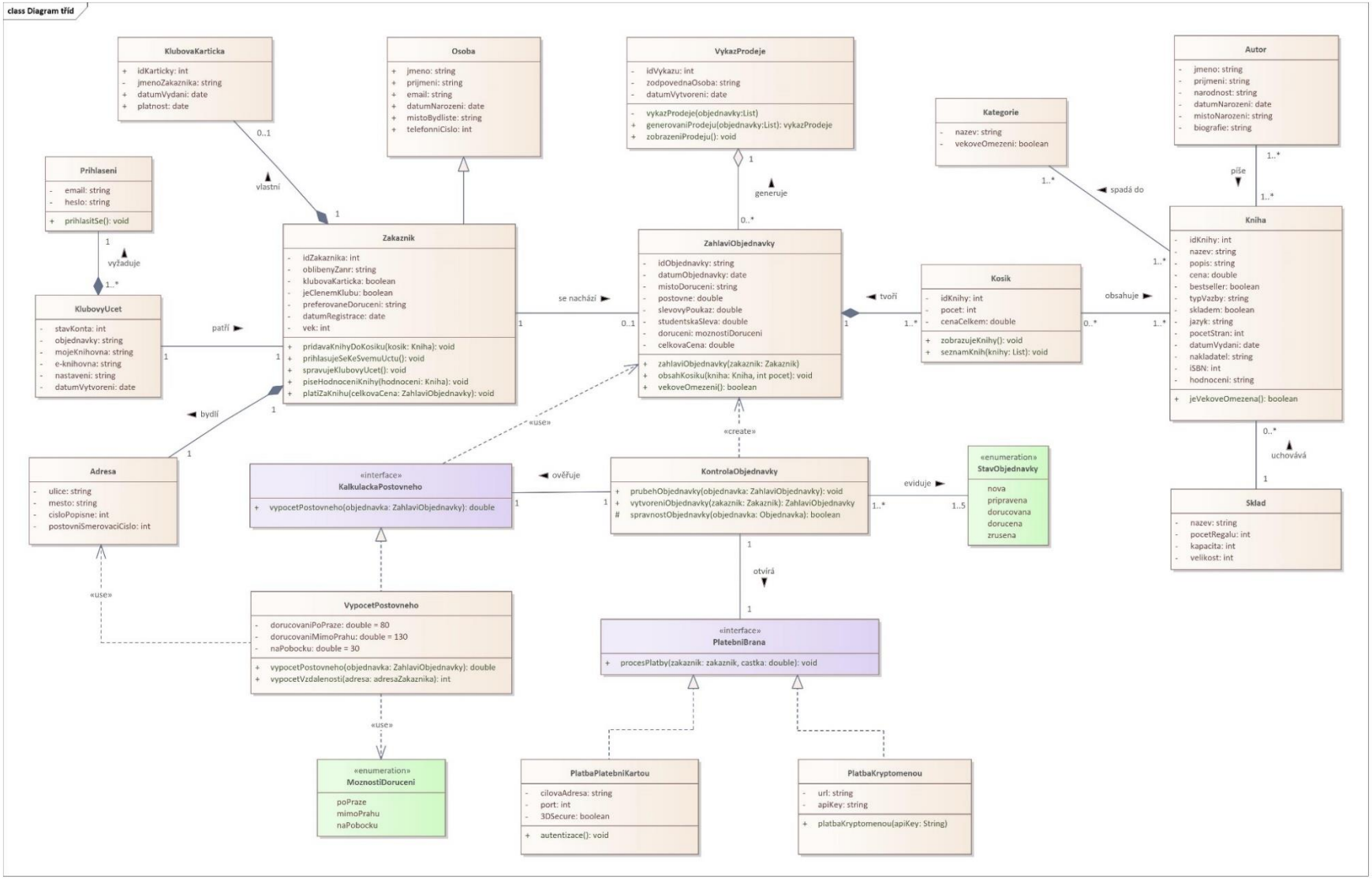
- jméno,
- příjmení,
- národnost,
- datum narození,
- místo narození,
- biografie.

Kategorie – druhou třídou, která se opět formou asociace váže ke třídě kniha, je třída kategorie, jež má následující atributy:

- název,
- věkové omezení.

Sklad – poslední třída, která úzce souvisí s knihou. Kromě toho, že slouží pro uskladnění knih, tak disponuje také určitou kapacitou nebo počtem regálů. V našem případě disponuje těmito atributy:

- název,
- počet regálů,
- kapacita,
- velikost.



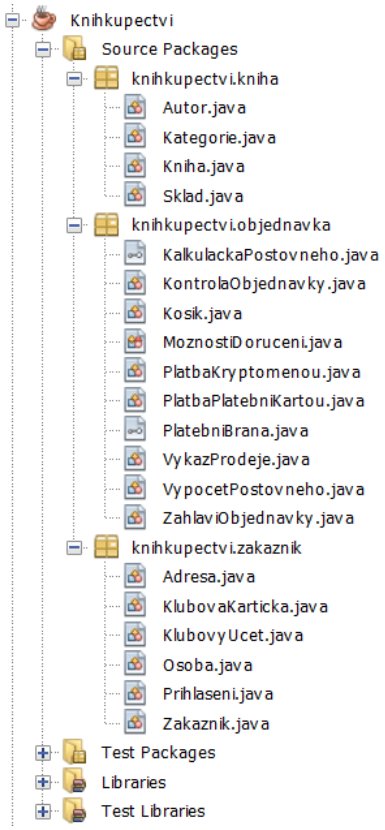
Obrázek 12 Návrhový model informačního systému, zdroj: vlastní

4.5 Transformace do jazyka Java

Na základě sestaveného návrhového modelu lze přistoupit k samotné transformaci do jazyka Java. K transformaci bude ve verzi 12.1. využit nástroj Apache Netbeans IDE pomocí kterého programátoři mohou psát, překládat, ladit a šířit programy. V Javě podporuje všechny 3 hlavní platformy - J2SE, J2EE a J2ME. Primárně je určeno pro vývoj aplikací v jazyce Java, ale může podporovat i další programovací jazyky. Vývojové prostředí NetBeans je bezplatně šířený produkt, který je možné používat bez jakýchkoliv omezení.

K tomu, abychom mohli přistoupit ke generování kódu z jazyka UML, potřebujeme nejprve nainstalovat plugin easyUML. Díky němu lze vytvořit totožný návrhový model obdobný tomu, který je na obrázku č. 12.

Plugin obsahuje všechny potřebné typy tříd a vazeb, které jsou nezbytné k vytvoření návrhového modelu. Jakmile je model vytvořený, klikneme na náš diagram tříd a pravým tlačítkem zvolíme možnost *„easyUML generate code“*. Vybereme Java project, do kterého chceme transformaci provést a klikneme na tlačítko *„Generate Code“*. Ve výsledku jsou třídy rozděleny do jednotlivých packages (zákazník, objednávka, kniha).



Obrázek 13 Rozdělení tříd do jednotlivých packages, zdroj: vlastní

4.5.1 Transformace z pohledu tříd

Víme, že použití jednotlivých typů tříd odpovídá požadavkům objektově orientovaných programovacích jazyků. Níže jsou uvedeny všechny typy, které byly v návrhovém modelu použity a jejich zobrazení v jazyce Java.

Třída – klíčovým spojením při generování kódu je *public class*. Kromě toho se vygenerují také příslušné atributy včetně jejich viditelnosti a datového typu. Příkladem třídy je adresa.

```
public class Adresa {

    private String ulice;
    private String mesto;
    private int cisloPopisne;
    private int postovniSmerovaciCislo;
```

Kód 1 Třída z pohledu transformace, zdroj: vlastní

Rozhraní – klíčovým spojením při generování kódu je *public interface*. Kromě toho se vygenerují také příslušné operace, včetně jejich názvu, parametrů a návratové hodnoty. Příkladem rozhraní je kalkulačka poštovného.

```
public interface KalkulackaPostovneho {

    public double vypocetPostovneho(ZahlaviObjednavky objednavka);
}
```

Kód 2 Rozhraní z pohledu transformace, zdroj: vlastní

Výčet (neboli enumerace) – klíčovým spojením při generování kódu je *public enum*. Kromě toho se vygenerují také všechny položky výčtu. Příkladem výčtu jsou možnosti doručení.

```
public enum MoznostiDoruceni {

    poPraze, mimoPrahu, naPobocku
}
```

Kód 3 Výčet z pohledu transformace, zdroj: vlastní

4.5.2 Transformace z pohledu relací

V návrhovém modelu bylo použito hned několik typů relací, které mezi předměty umožňují znázornit jejich vzájemnou komunikaci a vztahovost. Konkrétně byly použity vztahy asociace, agregace, kompozice, závislost, zobecnění a realizace, přičemž poslední dva zmíněné typy dokáže napřímo zahrnout také programovací jazyk Java.

Zobecnění – klíčovým slovem při generování kódu je *extends*. Ve vytvořeném modelu lze zobecnění najít mezi třídami zákazník a osoba, kdy zákazník dědí příslušné atributy od třídy osoba.

```
public class Zakaznik extends Osoba {

    private int idZakaznika;
    private String oblibenyZanr;
    private boolean klubovaKarticka;
    private boolean jeClenemKlubu;
    private String preferovaneDoruceni
    private Date datumRegistrace;
    private Adresa adresa;
```

Kód 4 Zobecnění z pohledu transformace, zdroj: vlastní

Důležitost tohoto typu relace lze vidět u metody *getVek()*, kdy pracujeme s proměnnou *datumNarozeni*, jež je součástí třídy Osoba. Kdyby třída zákazník nemohla dědit z třídy osoba, pak by proměnná musela být znovu deklarována.

```
public int getVek() {
    if (vek == 0) {
        Calendar birthCalendar = Calendar.getInstance();
        birthCalendar.setTime(datumNarozeni);

        Calendar now = Calendar.getInstance();

        vek = now.get(Calendar.YEAR) - birthCalendar.get(Calendar.YEAR);

    }
    return vek;
}
```

Kód 5 Příklad použití zobecnění, zdroj: vlastní

Realizace – klíčovým slovem při generování kódu je *implements*. Ve vytvořeném modelu lze realizaci najít mezi rozhraním a implementační třídou. Příklad lze nalézt mezi třídou výpočet poštovného a rozhraním kalkulačka poštovného.

```
public class VypocetPostovneho implements KalkulackaPostovneho {

    private static final double dorucovaniPoPraze = 80;
    private static final double dorucovaniMimoPrahu = 130;
    private static final double naPobocku = 30;
```

Kód 6 Realizace z pohledu transformace, zdroj: vlastní

4.5.3 Transformace z pohledu programátora

Důležitost výše uvedených transformací jednotlivých tříd a relací ocení každý programátor, jelikož nejenže dokáží ušetřit daný čas při psaní kódu, ale také mohou poskytnout lepší přehlednost nad celým modelem. Tímto ovšem práce programátora nekončí. Níže jsou uvedeny další kroky:

- **Import speciálních tříd** – u každého atributu je definován i jeho datový typ. Java si obecně poradí s primitivními datovými typy, jako je int, boolean apod. bez potřebné knihovny. Jinak je tomu ovšem u referenčních datových typů, se kterými se v paměti pracuje jinak. Příkladem může být datový typ date u proměnné *datumVytvoreni*. Abychom byli schopni definovat tento datový typ, musíme importovat package *java.time.LocalDate;*
- **Import packages** – v rámci jednotlivých tříd a packages lze narazit na situaci, kdy se budeme odkazovat na atribut z jiné třídy (package). Toto lze vyřešit importem dané package. Příkladem může být *knihkupectvi.zakaznik.Zakaznik;*
- **Gettery a settery** – metody k navrácení hodnoty (gettery) a pro zápis (settery) nejsou automaticky generovány. Naštěstí existuje způsob, jak si je jednoduše nechat vygenerovat. Stačí na privátní proměnnou kliknout pravým tlačítkem a zvolit *„Refactor“* a následně *„Encapsulate fields“*. Poté již pouze vybereme atributy, ke kterým chceme gettery a settery vygenerovat a zvolíme *„Refactor“*. Výsledek může vypadat např. takto:

```java
public boolean isJeClenemKlubu() {
    return jeClenemKlubu;
}

public void setJeClenemKlubu(boolean jeClenemKlubu) {
    this.jeClenemKlubu = jeClenemKlubu;
}

public String getPreferovaneDoruceni() {
    return preferovaneDoruceni;
}

public void setPreferovaneDoruceni(String preferovaneDoruceni) {
    this.preferovaneDoruceni = preferovaneDoruceni;
}

public LocalDate getDatumRegistrace() {
    return datumRegistrace;
}

public void setDatumRegistrace(LocalDate datumRegistrace) {
    this.datumRegistrace = datumRegistrace;
}
```

Kód 7 Gettery a settery z pohledu transformace, zdroj: vlastní

- **Operace (metody)** – v návrhovém modelu se u vybraných tříd vyskytují operace, které specifikují výsledek chování. Skrze generování lze získat viditelnost, jméno, základní hodnotu a návratový typ. Někdy ovšem operace mohou být ve výsledném kódu složitější. Příkladem může být metoda s názvem *prubehObjednavky*.

```java
public void prubehObjednavky (ZahlaviObjednavky objednavka) {
    if (!validaceObjednavky(objednavka)) {
        throw new IllegalStateException("Validace objednávky selhala - zákazník mladší 18 let nemůže zakoupit vybrané knihy");
    }

    double postovne = kalkulackaPostovneho.vypocetPostovneho(objednavka);
    objednavka.setPostovne(postovne);

    platebniBrana.procesPlatby(objednavka.getZakaznik(), objednavka.getCelkovaCena() + postovne);

}
```

Kód 8 Operace z pohledu transformace, zdroj: vlastní

5 Výsledky a diskuze

Výsledky této práce lze rozdělit do několika částí. V první řadě se povedl sběr požadavků formou interview od dvou renomovaných expertů, přičemž požadavky byly rozděleny dle základního dělení na funkční a nefunkční. U jednotlivých požadavků byla pečlivě stanovena jejich priorita, složitost, potažmo návaznost. Na základě získaných poznatků a požadavků bylo pomocí logického modelu přistoupeno k tvorbě návrhu informačního systému. Logický model byl rozdělen do 3 základních packages – zákazník, objednávka a kniha, přičemž každá package v sobě zahrnuje několik podstatných tříd. Kromě klasických tříd nalezneme v modelu také rozhraní či enumeraci. Komunikace a vztahovost mezi třídami je zajištěna skrze relace typu závislost, zobecnění, realizace či speciální typy asociace – agregace a kompozice. Důležité vlastnosti tříd zachycují atributy, zatímco výsledek chování operace. Výsledkem je kompletní návrhový model informačního systému.

Další část představuje transformaci modelu, kdy se podařilo transformovat jednotlivé třídy, včetně jejich atributů a operací do programovacího jazyka Java. Na transformaci lze nahlížet také z pohledu relací či čistě programátorského hlediska. Výsledkem jsou packages, které obsahují jednotlivé třídy. Každá třída byla ještě upravena tak, aby splňovala syntaxi a základní aspekty, které programovací jazyk Java vyžaduje.

Pokud vezmeme v potaz obsáhlost dnešních informačních systémů a komplexitu jazyka UML, dojde nám, že logický model je jen špičkou ledovce. Pokud bychom např. chtěli charakterizovat úplný seznam operací pro jednotlivé třídy, museli bychom pro vybrané třídy vytvořit stavové diagramy. Nefunkční požadavky bychom mohli dále rozdělit na business požadavky. Dalším aspektem by mohlo být přidání konkrétní podoby uživatelského rozhraní informačního systému. Existuje celá řada dalších postupů při návrhu a tvorbě IS, nicméně pro účely této práce byl největší důraz kladen na návrhový model a transformaci do jazyku Java.

6 Závěr

V teoretické části došlo nejprve k základní charakteristice klíčových pojmů, klasifikaci a členění informačních systémů. Následovaly kapitoly zaměřené na vývoj informačního systému z pohledu tradičních a agilních metodik, přičemž celý blok uzavřely jednotlivé fáze životního cyklu IS. Další kapitoly byly zaměřeny na jazyk UML, požadavky a programovací jazyk Java.

Cílem práce bylo za použití jazyka UML navrhnout diagram tříd, který bude reprezentovat statickou strukturu systému, znázorňovat datové struktury, operace u objektů a základní souvislosti mezi objekty. Návrh informačního systému byl v rámci případové studie realizován na knihkupectví, které se nachází v pražských Dejvicích. V praktické části nejprve formou interview s renomovanými experty došlo k definování funkčních a nefunkčních požadavků. Na základě požadavků došlo k rozdělení diagramu do jednotlivých packages, které můžeme chápat jako mechanismus UML pro sdružování prvků modelu na základě sémantické souvislosti. Každá package obsahuje několik podstatných tříd. V samotném diagramu byly použity různé typy relací a tříd. Výsledný diagram slouží pro další projektování informačního systému.

V rámci dílčího cíle došlo k transformaci do programovacího jazyka Java, přičemž největší důraz byl kladen na představení samotné transformace jednotlivých prvků či celých částí. Na transformaci bylo nahlíženo zejména z pohledu tříd, relací a samotného programátora. Výsledný kód v jazyce Java byl poté doplněn o příslušnou syntaxi, packages, speciální třídy, gettery, settery a základní metody.

Výsledný návrh informačního systému reflektuje základní potřeby a požadavky knihkupectví, díky čemuž je zajištěn především informační tok mezi zákazníkem a knihkupectvím. Dále lze provádět rychlejší a přesnější reporty či výstupy. Zmíněný návrh může též vést ke snížení celkových IT nákladů. V neposlední řadě knihkupectví díky tomuto návrhu získává nejednu konkurenční výhodu, jelikož tržní odvětví bývá mnohdy nepředvídatelné a čím dál více náročnější na rychlost, zpracování či kvalitu.

7 Seznam použitých zdrojů

[1] MOLNÁR, Zdeněk. *Moderní metody řízení informačních systémů*. Praha: Grada, 1992, 347 s. Nestůjte za dveřmi. ISBN 80-85623-07-2.

[2] VOŘÍŠEK, Jiří. *Principy a modely řízení podnikové informatiky*. Praha: Oeconomica, 2008, 446 s. ISBN 978-80-245-1440-6.

[3] DOUCEK, Petr a Richard BÉBR. *Manažerské informační systémy a jejich ekonomika*. Praha: Oeconomica, 2002, 92 s. ISBN 80-245-0412-X.

[4] SODOMKA, Petr a Hana KLČOVÁ. *Informační systémy v podnikové praxi*. 2., aktualiz. a rozš. vydání. Brno: Computer Press, a. s., 2010, 501 s. ISBN 978-80-251-2878-7.

[5] GÁLA, Libor, Jan POUR a Zuzana ŠEDIVÁ. *Podniková informatika*. 2., přeprac. a aktualiz. vydání. Praha: Grada, 2009, 496 s. ISBN 978-80-247-2615-1.

[6] ŠILEROVÁ, Edita. *Informační systémy v lesnictví* [online]. [cit. 2021-02-11]. Dostupné z: http://www.agris.cz/Content/files/main_files/62/140195/siler.pdf

[7] BASL, Josef a Roman BLAŽÍČEK. *Podnikové informační systémy*. 3., aktualizované a doplněné vydání. Praha: Grada, 2012, 328 s. ISBN 978-80-247-4307-3.

[8] DANEL, Roman. *Architektura informačních systémů podle úrovně řízení* [online]. 2011, , 5 [cit. 2021-02-15]. Dostupné z: https://homel.vsb.cz/~dan11/is_skripta/IS%202011%20-%20TPS-MIS-EIS.pdf

[9] PANEC, Zdeněk. Co je to Business intelligence?. *Systemonline.cz* [online]. 2003 [cit. 2021-02-15]. Dostupné z: https://www.systemonline.cz/clanky/co-je-to-business-intelligence.htm

[10] ZWETS, Berry. Does a standardized ERP system even exist?. *Techzine.eu* [online]. 2020 [cit. 2021-02-15]. Dostupné z: https://www.techzine.eu/blogs/cloud/47571/does-a-standardized-erp-system-even-exist/

[11] BRUCKNER, Tomáš. *Tvorba informačních systémů: principy, metodiky, architektury*. 1. Praha: Grada, 2012, 357 s. ISBN 978-80-247-4153-6.

[12] RÁČEK, Jaroslav. *Agilní metodiky vývoje SW* [online]. In: . 2013, s. 20 [cit. 2021-02-15]. Dostupné z: https://is.muni.cz/el/fi/podzim2013/PA017/um/SWE2_07_agilni.pdf

[13] ŠMÍD, Vladimír. *Životní cyklus informačního systému* [online]. [cit. 2021-02-15]. Dostupné z: https://www.fi.muni.cz/~smid/mis-zivcyk.htm

[14] VOŘÍŠEK, Jiří. *Strategické řízení informačního systému a systémová integrace*. Praha: Management Press, 1997, 324 s. ISBN 8085943409.

[15] KANISOVÁ, Hana a Miroslav MÜLLER. *UML srozumitelně*. 1. vyd. Brno: Computer Press, a. s., 2004, 176 s. ISBN 80-251-0231-9.

[16] ARLOW, Jim a Ila NEUSTADT. *UML a unifikovaný proces vývoje aplikací*. Brno: Computer Press, a. s., 2003, 408 s. ISBN 80-7226-947-X.

[17] TIŠŇOVSKÝ, Pavel. Nástroje pro tvorbu UML diagramů. *Root.cz* [online]. 2005 [cit. 2021-02-15]. Dostupné z: https://www.root.cz/clanky/nastroje-pro-tvorbu-uml-diagramu/#k01

[18] BOOCH, Grady, Ivar JACOBSON a James RUMBAUGH. *The unified modeling language user guide*. 2nd ed. Upper Saddle River: Addison-Wesley Professional, 2005, 494 s. ISBN 03-212-6797-4.

[19] URBAN, Vít. *E-learningové materiály pro výuku jazyka UML*. Brno, 2013. Dostupné také z: https://is.muni.cz/th/tyzvv/BP_urban.pdf. Bakalářská práce. Masarykova univerzita.

[20] VRANA, Ivan. *Projektování informačních systémů s UML*. 1. vyd. Praha: Česká zemědělská univerzita v Praze, 2008, 147 s. ISBN 978-80-213-1817-5.

[21] FOWLER, Martin. *UML distilled: a brief guide to the standard object modeling language*. 3rd ed. Boston: Addison-Wesley Professional, 2004, 175 s. ISBN 03-211-9368-7.

[22] SAKAIYA, Taichi. Design Goals of the Java Programming Language. *Oracle.com* [online]. [cit. 2021-02-15]. Dostupné z: https://www.oracle.com/java/technologies/introduction-to-java.html

[23] SCHILDT, Herbert. *Java 8: Výukový kurz*. Brno: Computer Press, a. s., 2016, 396 s. ISBN 978-80-251-4665-1.

[24] *Život objektu* [online]. In: . [cit. 2021-02-15]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=418

[25] Java - rozhraní, polymorfismus, výjimky. *Portal.haka-software.cz* [online]. [cit. 2021-02-15]. Dostupné z: http://portal.haka-software.cz/index.php?option=com_content&view=article&id=9:java-rozhrani-polymorfismus-vyjimky&catid=3:programovani-java&Itemid=13

[26] FAIGL, Jan. *Objektově orientované programování* [online]. In: . [cit. 2021-02-15]. Dostupné z: https://cw.fel.cvut.cz/old/_media/courses/a0b36pr1/lectures/lecture09-handout-2x2.pdf

8 Přílohy

Seznam příloh na CD:

- zdrojový kód k návrhovému modelu,
- diagram tříd a packages.