
**Slezská univerzita v Opavě**
**Obchodně podnikatelská fakulta v Karviné**

# OBJEKTOVÉ METODY MODELOVÁNÍ

## *VÝKLAD JAZYKA UML, JEHO MOŽNOSTI A PRAKTICKÉ UKÁZKY*

Pro prezenční formu studia

**Zdeněk Franěk**

**Karviná 2014**

Projekt OP VK č. CZ.1.07/2.2.00/28.0017
„Inovace studijních programů na Slezské univerzitě,
Obchodně podnikatelské fakultě v Karviné“

**Obor:** Informatika.

**Anotace:** Tento učební text se zabývá metodami návrhu informačních systémů s využitím metodiky RUP a UML diagramů. Je určen pro studenty 1. ročníku v navazujícím stupni studia oboru Manažerská informatika, studijního programu Systémové inženýrství a informatika. Hlavním tématem studijní opory je výklad jazyka UML, předvedení jeho možností a řada ukázek jeho užití v praxi. Modelovací jazyk UML (Unified Modeling Language) je základní modelovací nástroj pro objektovou analýzu a návrh software, obecně informačních systémů.

**Klíčová slova:** Modelování IS, diagramy UML, metodika RUP, objekty a třídy, polymorfismus.



**Autor:** **RNDr. Zdeněk Franěk, Ph.D.**

**Recenzenti:** Doplňte jména a příjmení včetně titulů

**ISBN** Doplní oddělení vědy a výzkumu.

# OBSAH


**ÚVOD ..... 6**

**1 OBJEKTOVÉHO MODELOVÁNÍ ..... 7**

1.1 OD STRUKTURÁLNÍHO POJETÍ K OBJEKTOVÉMU ..... 7
1.2 OBJEKT ..... 8
1.3 OO - DALŠÍ VLASTNOSTI ..... 10

*1.3.1 TŘÍDA ..... 10*
*1.3.2 ZOBECNĚNÍ (GENERALIZACE-SPECIALIZACE), DĚDĚNÍ (INHERITANCE) ..... 11*
*1.3.3 PŘEKRÝVÁNÍ METOD ..... 11*
*1.3.4 ABSTRAKTNÍ METODY A TŘÍDY ..... 12*
*1.3.5 POLYMORFISMUS ..... 12*

**2 PRINCIP A ZÁKLADNÍ ELEMENTY JAZYKA UML ..... 14**

2.1 HISTORIE : ..... 15
2.2 UML - SKLADBA ..... 16
2.3 UML - MECHANISMY ..... 17

**3 POPIS JAZYKA UML – PŘÍPADY UŽITÍ ..... 19**

3.1 PŘÍPADOVÁ STUDIE¨ ..... 19

*3.1.1 PŘÍJEM ZAKÁZKY NA OPRAVU SPOTŘEBIČE ..... 20*
*3.1.2 VYDEJ ZAKÁZKY MAJITELI: ..... 20*
*3.1.3 SPRÁVA ČÍSELNÍKŮ APLIKACE ..... 21*

3.2 AKTÉŘI ..... 21
3.3 PŘÍPADY UŽITÍ ..... 22
3.4 POPIS PŘÍPADU UŽITÍ ..... 24
3.5 VSTUPNÍ A VÝSTUPNÍ PODMÍNKY ..... 25
3.6 POSTUP TVORBY POPISU PŘÍPADU UŽITÍ ..... 26

**4 POPIS JAZYKA UML – MODELOVÁNÍ TŘÍD A OBJEKTŮ ..... 27**

4.1 ZÁKLADNÍ POPIS DIAGRAMU TŘÍD ..... 27
4.2 PRVKY DIAGRAMU TŘÍD ..... 28
4.3 DOPORUČENÍ K VYTVÁŘENÍ MODELU TŘÍD ..... 28
4.4 PŘÍKLADOVÝ MODEL TŘÍD ..... 29

**5 POPIS JAZYKA UML – MODEL OBJEKTOVÉ SPOLUPRÁCE ..... 31**

5.1 ZÁKLADNÍ CHARAKTERISTIKA OBJEKTOVÉHO DIAGRAMU ..... 31
5.2 PRVKY OBJEKTOVÉHO DIAGRAMU ..... 31
5.3 DOPORUČENÍ K VYTVÁŘENÍ OBJEKTOVÉHO DIAGRAMU ..... 31
5.4 PŘÍKLADOVÝ OBJEKTOVÝ DIAGRAM ..... 32

**6 POPIS JAZYKA UML – STAVOVÉ DIAGRAMY ..... 33**

6.1 ZÁKLADNÍ CHARAKTERISTIKA ..... 33
6.2 PRVKY STAVOVÉHO DIAGRAMU ..... 33
6.3 DOPORUČENÍ K VYTVÁŘENÍ STAVOVÉHO DIAGRAMU ..... 33
6.4 PŘÍKLADOVÝ STAVOVÝ DIAGRAM ..... 34

**7 POPIS JAZYKA UML – DIAGRAMY AKTIVIT ..... 36**

7.1 ZÁKLADNÍ CHARAKTERISTIKA DIAGRAMU AKTIVIT ..... 36
7.2 PRVKY DIAGRAMU AKTIVIT ..... 36
7.3 DOPORUČENÍ K VYTVÁŘENÍ DIAGRAMU AKTIVIT ..... 37
7.4 PŘÍKLADOVÝ DIAGRAM AKTIVIT ..... 38

**8 SEKVENČNÍ DIAGRAM ..... 42**

8.1 ZÁKLADNÍ CHARAKTERISTIKA SEKVENČNÍHO DIAGRAMU ..... 42
8.2 DOPORUČENÍ K VYTVÁŘENÍ SEKVENČNÍHO DIAGRAMU ..... 42
8.3 PŘÍKLADOVÝ SEKVENČNÍ DIAGRAM ..... 43

**9 PŘEHLED SOFTWARE PRODUKTŮ PRO PRÁCI S UML ..... 45**

9.1 IBM RATIONAL SOFTWARE DEVELOPMENT PLATFORM ..... 45
9.2 ENTERPRISE ARCHITECT FIRMY SPARX ..... 47
9.3 UML A VISIO FIRMY MICROSOFT ..... 47

**10 RUP - RATIONAL UNIFIED PROCESS ..... 49**

10.1 HISTORIE ..... 49
10.2 ZÁKLADNÍ PRAVIDLA RUP ..... 50

*10.2.1 ITERATIVNÍ VÝVOJ ..... 50*
*10.2.2 AKTIVNÍ SPRÁVA POŽADAVKŮ ..... 51*
*10.2.3 KOMPONENTOVÁ ARCHITEKTURA ..... 51*
*10.2.4 VIZUÁLNÍ MODELOVÁNÍ ..... 52*
*10.2.5 OVĚŘOVÁNÍ KVALITY SOFTWARE ..... 52*
*10.2.6 ŘÍZENÍ ZMĚN ..... 52*

10.3 ŽIVOTNÍ CYKLUS PROJEKTU: CHARAKTERISTIKA JEDNOTLIVÝCH FÁZÍ ..... 52

*10.3.1 ŽIVOTNÍ CYKLUS PRODUKTU V RUP ..... 52*

10.4 ZAHÁJENÍ (INCEPTION) ..... 53
10.5 PŘÍPRAVA (ELABORATION) ..... 55
10.6 KONSTRUKCE (CONSTRUCTION) ..... 57
10.7 PŘEDÁVÁNÍ (TRANSITION) ..... 59
10.8 DISCIPLINY SPOJENÉ S PROJEKTEM ..... 61

*10.8.1 VYSVĚTLENÍ POUŽITÝCH SYMBOLŮ ..... 61*
*10.8.2 DISCIPLINY V RUP ..... 61*
*10.8.3 TVORBA PODNIKOVÉHO MODELU ..... 62*
*10.8.4 SPRÁVA POŽADAVKŮ ..... 63*
*10.8.5 ANALÝZA A NÁVRH ..... 65*
*10.8.6 IMPLEMENTACE ..... 67*
*10.8.7 TESTOVÁNÍ ..... 68*
*10.8.8 NASAZENÍ ..... 69*
*10.8.9 ŘÍZENÍ PROJEKTU ..... 71*
*10.8.10 ŘÍZENÍ ZMĚN A KONFIGURACE ..... 72*
*10.8.11 SPRÁVA PROSTŘEDÍ ..... 74*

10.9 OSOBY PODÍLEJÍCÍ SE NA PROJEKTU (ROLE) ..... 76

*10.9.1 ŘÍZENÍ PROJEKTU ..... 76*
*10.9.2 TVORBA PODNIKOVÉHO MODELU ..... 76*
*10.9.3 SPRÁVA POŽADAVKŮ ..... 76*
*10.9.4 SPRÁVA PROSTŘEDÍ ..... 76*
*10.9.5 ANALÝZA A NÁVRH ..... 77*
*10.9.6 TESTOVÁNÍ ..... 77*
*10.9.7 IMPLEMENTACE ..... 77*
*10.9.8 NASAZENÍ ..... 78*

**11 PRAKTICKÉ PŘÍKLADY VYUŽITÍ UML ..... 79**

**ZÁVĚR ..... 110**

**SEZNAM POUŽITÉ LITERATURY ..... 111**

11.1 INTERNETOVÉ ZDROJE: ..... 112
11.2 SEZNAM UKÁZKOVÝCH SEMINÁRNÍCH PRACÍ: ..... 112
**PŘÍLOHA Č. 1 SEZNAM OBRÁZKŮ ..... 113**
**PŘÍLOHA Č. 2 STRUKTURA UML ..... 114**

# ÚVOD

Předmět „Objektové metody modelování“ je součástí inovovaného studijního plánu studijního programu „Systémové inženýrství a informatika“, oboru „Manažerská informatika“, který garantuje „Katedra informatiky a matematiky“, Slezské univerzity v Opavě, Obchodně podnikatelské fakulty v Karviné. Tento předmět je nabízen jako povinně volitelný pro 1. ročník navazujícího magisterského stupně studia.

Uplynulých pět let bylo toto téma vyučováno v předmětu s názvem „Úvod do objektového modelování“. V roce 2012 byla osnova předmětu a jeho náplň inovována. Tento učební text sleduje inovovanou osnovu předmětu a nabízí studentům souhrnný text s výkladem teorie i praxe metod objektového modelování.

Hlavním tématem studijní opory je výklad jazyka UML, předvedení jeho možností a řada ukázek jeho užití v praxi. Modelovací jazyk UML (Unified Modeling Language) je základní modelovací nástroj pro objektovou analýzu a návrh software, obecně informačních systémů.

Souhrn metod modelovacího jazyka UML až do aktuálně používané verze 2.0 se stal průmyslovým standardem. Modelovací jazyk UML je souhrnem zejména grafických notací k vyjádření analytických a návrhových modelů při vytváření software. Je to jazyk, který umožňuje modelovat jednoduché i složité aplikace pomocí jednotné formální syntaxe. Umožňuje tak sdílet výsledky při vývoji software v týmové práci. Vybrané diagramy jsou určeny rovněž pro porozumění mezi zadavateli a uživateli aplikace a tvůrci software. UML verze 2.0 je jazyk pro vizualizaci, specifikaci, stavbu a dokumentaci softwarových produktů.

Nasazení UML má širší souvislosti. Postupy popisované pomocí diagramů jazyka UML jsou součástí obecné metodiky v procesu vývoje software běžně označován jako Unified Process. Je to ověřená metoda vývoje software, a proto je jí v tomto učebním textu věnována samostatná kapitola. Objasní se tak provázanost metodiky s nástroji (diagramy) jazyka UML.

Pro realizaci (kreslení) diagramů UML existuje v současné době řada software produktů. V rámci předmětu jsou nejpoužívanější software nástroje předvedeny a účastníky využity při řešení konkrétních problémů. V učebním textu budou vybrané software nástroje stručně popsány s cílem dát studentům návod pro výběr a použití konkrétního nástroje při tvorbě seminární práce. Jedná se o tyto produkty: CASE nástroj firmy IBM, Enterprise architect firmy Sparkx a UML diagramy obsažené ve VISIO firmy Microsoft. V současnosti všechny ve světě rozšířené objektově orientované nástroje CASE (Computer Aided Software Engineering – nástroje pro podporu analýzy a návrhu aplikací) vychází z modelovacího jazyka UML.

Analytické práce s využitím výše uvedené metodiky a jazyka UML samozřejmě mají pokračování v samotném programování, tedy využitím programovacích jazyků zahrnující v sobě objektově orientované metody. Cílem této učebnice není detailně popsat objektově orientované techniky programování.

Samozřejmě objektově orientované modelovací techniky mají své uplatnění a efektivitu při programování v software nástrojích Eclipse, NetBeans, programovací jazyky C## a Java Rozsah výkladu programovacích technik přesahuje rámec této učební pomůcky, přesto v textu jsou uvedeny ilustrativní příklady, bez nichž by výklad objektově orientovaných technik návrhu software byl ochuzen. Předmět má logickou provázanost na předmět objektové programování, ve kterém jsou probrány programátorské techniky C## a Java.

Hlavním cílem této učební opory je seznámení se všemi hlavními diagramy UML a poskytnout zdroj rozumného výkladu usnadňující pochopení UML 2.0. Znalost této problematiky je nezbytným vybavením inženýra oboru manažerská informatika.

# 1 OBJEKTOVÉHO MODELOVÁNÍ

Tato kapitola seznamuje studenty se základními pojmy objektově orientovaného návrhu software.

## 1.1 OD STRUKTURÁLNÍHO POJETÍ K OBJEKTOVÉMU

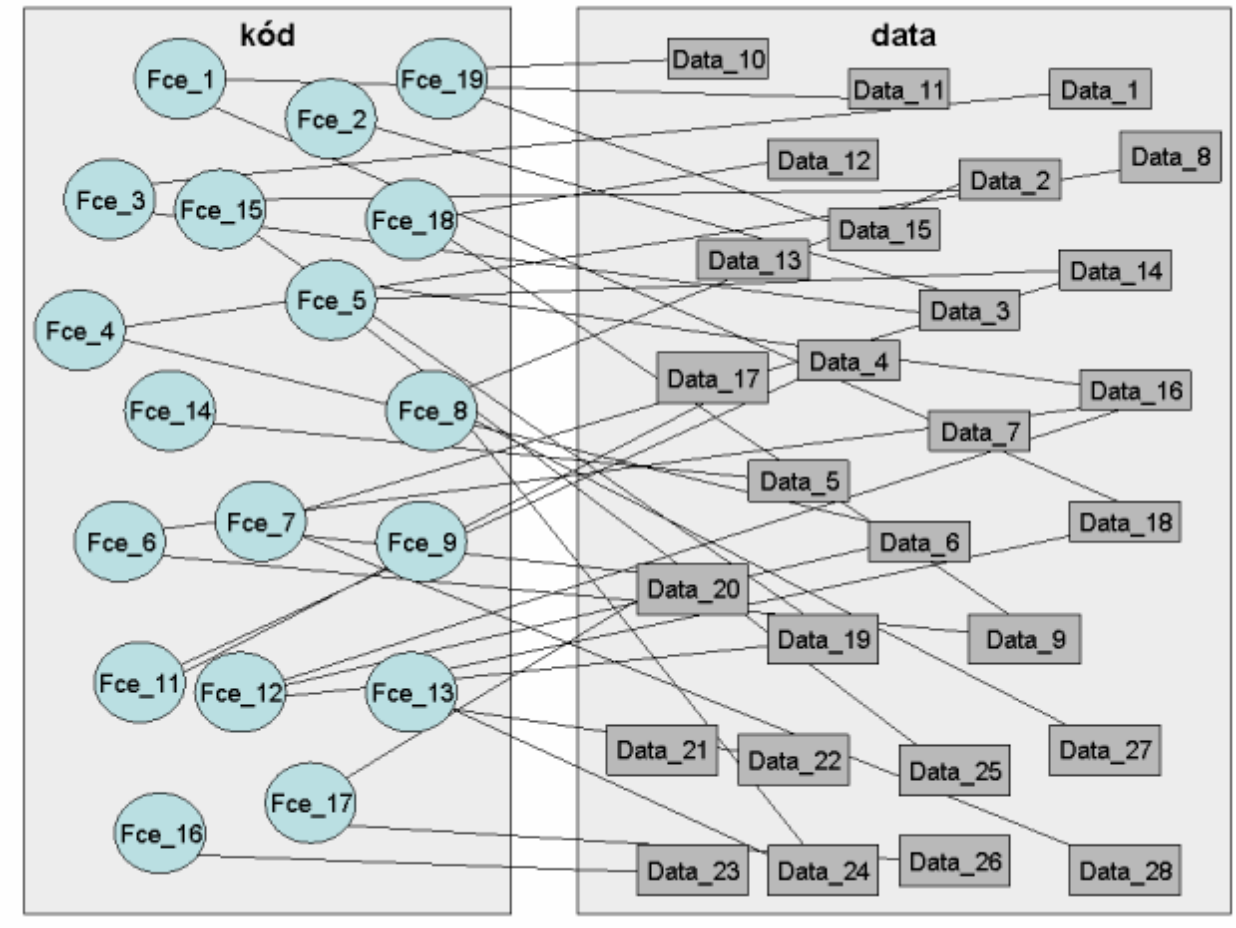
Strukturální pojetí bylo charakteristické tím, že kód programu pracuje s daty.

**Definujme si kód:** tyto prvky čtou a mění data; jsou to výkonné části informačního systému, provádějí nějakou činnost; mohou to být např. binární programy (či jejich části - moduly, funkce, procedury, ...), skripty, triggery.

**Definujme si data:** jak *proměnné* (lokální či globální) držené jen v paměti (po vypnutí počítače se jejich obsah, ale vlastně i jejich samotná existence "ztratí"), tak *perzistentní data* (soubory, řádky db tabulek, apod.)

Charakteristický obrázek - část systému, kde kód pracuje s daty:

**Obrázek 1:** Strukturální pojetí - kód a data




Zdroj: http://mpavus.wz.cz

Tato situace přináší různé problémy - ukážeme si některé z nich (tj. ty, s jejichž řešením nám může pomoci objektový přístup).

Při vývoji IS musíme provádět **transformaci**: nestačí nám prostě namodelovat realitu, ale navíc ji musíme *rozdělit* na oblast dat a na oblast kódu; již tato transformace nás stojí jisté úsilí, ba co víc: důsledky této transformace se "táhnou" celým dalším vývojem a údržbou IS.

Toto *rozdělení* nám ve fázi údržby a dalšího rozvoje může způsobit daleko větší obtíže, než které jsme si prožili při prvotním vytvoření IS.

Vznikají problémy při údržbě a dalším rozvoji IS pokud chceme přidat/změnit funkcionalitu: jak jednoduše zjistit, která **data** jsou ovlivněna kterou **funkcí**? Konkrétní funkce může volat další funkce, ty mohou volat opět další funkce, a až v těchto volaných funkcích může být čtena/měněna určitá oblast dat (a co hůř - toto čtení/měnění může být podmíněné, tj. může se provádět jen za určitých podmínek, jejichž vyhodnocení může být samo o sobě velmi složité a může být závislé na dalších funkcích a na hodnotě dalších dat; hodnota, na kterou je měněna nějaká oblast dat, může být také nastavena dle výsledku volání dalších funkcí a dle hodnoty dalších dat). Dalším problém při údržbě a dalším rozvoji: jak jednoduše zjistit, kterak spolu souvisí různá data, jaké vztahy mezi daty panují, co musí být dodrženo pro to, aby obsah dat byl validní?

Další obtíže vznikají při ladění a testování kódu: funkce spolu sdílí (neperzistentní) data v podstatě dvěma způsoby:

1. lokální data: ta musí být předána jako parametr
2. globální data: jsou viditelná a měnitelná i bez jejich předávání

V souvislosti s tím vznikají další otázky, jak však ladit (a testovat) jednotlivé funkce, nebo jednotlivé "situace", jež mohou nastat?, jak zajistit, aby při ladění určité funkce byly správně nastaveny hodnoty všech globálních proměnných, a aby byla funkce zavolána se správnými hodnotami všech parametrů (je nutno nejen nastavit smysluplné hodnoty parametrů a globálních proměnných, ale musíme mít na zřeteli i správnou kombinaci všech hodnot, která nastane právě při testované situaci)?

A další obtíží je znovu-použitelnost (re-use).

Závažnost výše uvedených problémů a obtížnost jejich řešení může velmi vzrůst za těchto situací:

- vývoj IS je rozprostřen v čase (tj. práce na vývoji IS trvají *dlouho*)
- vývoj IS je rozprostřen v prostoru (tj. členové týmu nesedí spolu v jedné místnosti, nebo dokonce ani ve stejné budově či ve stejném městě, anebo se na vývoji podílí více řešitelských firem, či firma řešitelská i zadavatelská)
- vyšší fluktuace řešitelů (tím je myšleno nejenže zaměstnanec odejde z řešitelské firmy, ale i situace kdy je vývojář např. "převelen" na jiné úkoly a musí ho zastoupit někdo jiný)
- větší řešitelské týmy
- reinženýring (myšleno: systém či jeho část vyvineme znovu - např. z důvodu přechodu na jinou technologii, či systém nabobtnal tak, že jeho další údržba a rozvoj se jeví jako *dražší* než *zahodit ho a udělat to znovu a lépe*,….)

## 1.2 OBJEKT

Pan Booch to vyjádřil takto: objekt má stav, chování, a identitu; struktura a chování podobných objektů je definována v jejich obecné třídě; termíny instance a objekt jsou zaměnitelné. Objekt je **uzavřený**, lokální data jsou *obalena* metodami, ale ani metody nejsou

zvenčí přístupné - jediný způsob, jak použít objekt či jak s ním komunikovat, je **poslat objektu zprávu**: pak objekt spustí svou metodu, která může použít atributy objektu a další metody objektu, může také poslat zprávu jinému objektu.

Podrobnější popis:

- **protokol zpráv**: seznam zpráv, které je možno objektu poslat; je to to jediné, co *je vidět* z okolí objektu - všechno ostatní je uzavřeno uvnitř objektu a není to z okolí objektu přístupné
- **metody (chování)**: po té, co objekt obdrží zprávu, spustí odpovídající metodu, tj. posloupnost kódu vykonávající nějakou činnost; metoda používá atributy objektu a další metody objektu, také může poslat zprávu jinému objektu; metody jsou zvnějšku neviditelné a nepřístupné
- **atributy (lokální data)**: je to vnitřní paměť objektu; stav objektu je určen právě hodnotami těchto atributů; atributy jsou zvnějšku neviditelné a nepřístupné
- **odkaz na jiný objekt**: pokud objekt *zná* odkaz na jiný objekt, může mu poslat zprávu; zpráva je poslána příkazem umístěným v kódu metody

Podrobněji o posílání zpráv:

- **vysílač, sender, klient**: objekt, který posílá jinému objektu zprávu
- **příjemce, receiver, server**: objekt, kterému je poslána zpráva; příjemce má přiřazeny jednotlivé zprávy k jednotlivým metodám - po přijetí zprávy tedy příjemce spustí odpovídající metodu
- **zpráva** si sebou nese parametry (ty jsou předány odpovídající metodě), po skončení metody příjemce vrátí návratovou hodnotu zpět vysílači

Dva příklady k *objektovému myšlení*:

- **Objekt je black-box**: mějme černou skříňku s několika tlačítky a světýlky. Může to být TV, automobil, PC, šachový počítač, mikrovlnka. Co potřebujeme znát pro použití tohoto objektu je: **jaké chování má tento objekt**? Jaké tlačítko mám použít, aby se objekt zachoval tak, jak právě potřebuji (mikrovlnka: abych nastavil dobu a stupeň ohřevu, abych namísto ohřevu spustil grilování,…. TV: abych zapnul TV, abych pod konkrétní číslo kanálu uložil konkrétní číslo programu, abych zesílil zvuk). Je mi naprosto lhostejno *co je uvnitř*, tj. pokud požádám TV o zesvětlení obrazu, TV toto provede (TV vnitřně použije jednu či více metod, popř. interně zavolá o služby další objekty, které jsou *zabaleny* uvnitř v TV); anebo: nevím nic o tom, na základě jakých zapamatovaných údajů si mikrovlnka v závislosti na váze a druhu zmražené potraviny vypočítá dobu a výkon k rozmražení - a přesto jsem schopen mikrovlnku používat.
- **Objekt jako HW:** komponenty dnešních PC mají výše uvedené objektové vlastnosti, jednotlivé komponenty (např. zvuková karta, pevný disk, CD-ROM, ...) se chovají jako objekty:
    - po správném zapojení a zprovoznění tyto *objekty* spolu spolupracují: jeden objekt požaduje po jiném objektu (resp. počítačová komponenta žádá jinou komponentu o) provedení služeb, které se *zavázal* tento objekt poskytovat - spoluprací všech objektů je zajištěna správná činnost systému (např. komponenty PC spolupracují - je to fungující PC!

- komponenty mají vnitřní paměť - kromě dat z *problémové domény* (například u TV karty TV signál, video a audio signál) obsahují nepochybně další vnitřní paměť, o které ostatním komponentám není nic známo, a nejsou pro ostatní komponenty nijak dosažitelné
- komponenty mají vnitřní metody, kterými řídí svou činnost - tyto metody jsou však ostatním komponentám také nedosažitelné (některé z nich lze však spustit tím, že je objektu zaslána odpovídající zpráva)
- komponenty lze skládat a tvořit tak složené objekty: složením základní desky, CD-ROM, pevného disku, zvukové karty, paměťového bloku, atd. atd. dostaneme PC, které má opět charakter objektu - propojíme-li správně několik PC tak, že dostaneme počítačovou síť, pak si mohou jednotlivé PC posílat zprávy (a přitom jednotlivá PC neví nic o tom, z jakých komponent se skládají ostatní PC - jedno PC může jinému PC pouze poslat zprávu); v opačném směru to platí také - základní deska obsahuje několik integrovaným obvodů, které se chovají také jako objekty: posílají si zprávy, a pokud je objekt požádán o službu, tak ji provede - vzájemnou kooperací těchto objektů (IO) plní základní deska očekávané funkčnosti a mohou ji používat další objekty (další komponenty PC)

## 1.3 OO - DALŠÍ VLASTNOSTI

Musíme si uvést další vlastnosti objektové orientace. Tyto vlastnosti jsou odvozené, všechny vyplývají z výše uvedené definice objektu, ale velmi nám usnadňují vývoj OO systému – v podstatě se bez těchto vlastností dnes již neobejdeme.

### *1.3.1 TŘÍDA*

**Třída reprezentuje šablonu pro objekty a popisuje interní strukturu těchto objektů.**

Mohli bychom všechny objekty definovat *přímo* v kódu (tj. tam, kde by bylo potřeba vzniku nového objektu, by byl napsán kód definující všechny atributy i metody objektu). Pokud ale budeme mít více objektů, které budou mít stejné atributy a metody, jsme nuceni znovu a znovu definovat objekty stejného typu. Ovšem máme mechanismus umožňující re-use (znovu-použitelnost již jednou *udělaného*) :

nadefinujeme třídu jako vzor, šablonu či předpis pro vznik budoucích objektů. Vytvoříme tedy třídu xxxx a při následné potřebě vzniku nového objektu pak namísto jeho plného popisu jen řekneme:

vytvoř objekt bbbbb , který je typu xxxxx.

**Příklad:** V informačním systému evidujícím veterináře mějme třídu **Veterinář** s atributy uchovávajícími informace o veterináři (jméno, titul atd.) a se dvěma metodami - ty umí ověřit členství veterináře v komoře veterinárních lékařů a odeslat veterináři e-mail. Na následujícím obrázku je znázorněna třída **Veterinář** se dvěma vytvořenými objekty **nový** veterinář (právě zaváděný do systému) a **hlavní** veterinář ČR (tedy nevím, jestli taková funkce v ČR vůbec existuje.

**Poznámka:** pro tyto obrázky budu používat notaci UML (i když jsme ji ještě neprobrali). V této notaci má třída tvar obdélníku, kde v horním bloku je název třídy, v prostředním bloku pak může být uveden seznam atributů a v dolním seznam metod. Podobně

je zobrazen i objekt, který má konkrétní hodnoty atributů a jeho jméno je podtrženo. Stereotypem <<instantiate>> u závislosti (znázorněné šipkou s přerušovanou čárou) mezi objektem a třídou je znázorněno, že objekt závisí na třídě tím způsobem, že objekt vzniká jako instance této třídy.

### *1.3.2 ZOBECNĚNÍ (GENERALIZACE-SPECIALIZACE), DĚDĚNÍ (INHERITANCE)*

**Zobecnění nám umožňuje další stupeň re-use : znovu-použitelnost ne ve vztahu třída a několik jejích instancí, ale třída a od ní několik odvozených tříd.**

Pokud namodelujeme několik tříd, které jsou si velmi *podobné*, mohlo by se jednat o situaci, kde je vhodné použít zobecnění.

**Příklad:** Klasický příklad pro znázornění zobecnění: obecnější třída (**předek, nadtřída, bázová třída, předchůdce**) se jmenuje **tvar** a zastupuje *nějaký, blíže neurčený* dvourozměrný tvar. Tvar má svou barvu čáry, barvu výplně, pozici (pozice těžiště), šířku a výšku, dále má metody **nakresli** (nakreslí tvar), **spočítejObsah** (spočítá plošný obsah) a **spočítejPlochuOhraničení** (spočítá plochu pravoúhelníku ohraničujícího tvar).

**Podtřídy (potomci)** jsou specializacemi **nadtřídy** -zde máme **kruh, pravoúhelník, trojúhelník**. Všude (tj. v jiných diagramech, v kódu, ....), kde použijeme instanci (objekt) třídy **tvar** můžeme použít instanci libovolného jeho potomka. Říkáme, že "potomek je druhem předka" tj. **kruh je druhem tvaru**, **trojúhelník je druhem tvaru** - pokud si tuto větu říct nemůžeme, pravděpodobně je ve stromu zobecnění nějaká chyba (opakovaná chyba v učebnicích: předek je bod, potomek je úsečka, přímka, čtverec, kruh, ... - zde si nemůžeme říct *čtverec je druhem bodu*). Vztahy generalizace-specializace se v konkrétním prostředí realizují pomocí techniky **dědění (inheritance)**: potomci dědí:

- atributy,
- metody,
- relace,
- omezení.

Potomci mohou ke zděděnému přidávat *to svoje* (tj. atributy, metody, relace a omezení), dokonce můžou zděděné metody předefinovat (viz níže výklad pro abstraktní metody). Tj. kruh, pravoúhelník a čtyřúhelník zdědí od předka **tvar** všechny atributy (barvu čáry, šířku a výšku, ...), dále zdědí metodu (a její kód) spočítejPlochuOhraničení() a předefinují metody nakresli() a spočítejObsah().

### *1.3.3 PŘEKRÝVÁNÍ METOD*

**Potřebuje-li potomek používat jiný kód zděděné metody, pak ho může překrýt svým kódem (ale signaturu metody musí dodržet).**

**Příklad:** Ve výše uvedeném obrázku je metoda spočítejPlochuOhraničení() zděděna potomky a funguje (je univerzální, její kód je platný pro všechny potomky, tedy ji nadefinujeme jen jednou v bázové třídě). Ovšem metoda spočítejObsah() je pro každého potomka jiná - jinak budeme počítat plochu pravoúhlého trojúhelníka, jinak trojúhelníka a jinak kruhu. Metoda spočítejObsah() tedy nemůže být definována jen jednou v bázové třídě - musíme ji definovat pro každého speciálního potomka znovu. Podobně je to s metodou nakresli() - každý speciální tvar bude kreslen jinak, nelze vytvořit obecný algoritmus platný pro všechny potomky.

Tyto skutečnosti ovšem nedávají odpověď na otázku: proč tedy vůbec uvádíme metody spočítejObsah() a nakresli() i v bázové třídě? Nebylo by jednodušší v bázové třídě uvést jen metodu spočítejPlochuOhraničení() a zbylé dvě metody vynechat - o zavedení metod nechť se postarají potomci? Existuje několik důvodů, proč uvádíme metody v bázové třídě a pak je v potomcích překrýváme:

- můžeme mít situaci, kdy většina potomků zdědí metodu od svého předka, přičemž je metoda beze změny použitelná i v potomcích, ale máme pár *výjimečných* potomků, kde algoritmus v metodě nevyhovuje - tedy jen v těchto výjimečných potomcích potřebujeme metodu překrýt
- může se stát, že když vytváříme strom inheritancí, tak ještě nevíme, které všechny potomky budeme mít nakonec ve stromu (nebo to už víme, ale ještě přesně nevíme, jak se bude potomek chovat), a tudíž ještě není jisté, zda u všech potomků bude moci být použita metoda tak jak je nadefinovaná v předkovi: nadefinujeme tedy metodu už v předkovi, a jak postupně upřesňujeme strom inheritancí, tak metodu v potomkovi buď *ponecháme tu zděděnou*, nebo ji kdykoliv podle potřeby předefinujeme. Toto má větší význam než by se mohlo zdát - a to při budoucím vývoji systému a údržbě: snadno tím můžeme upravovat strom inheritancí, jen ve všech potomcích musíme zajistit dodržení signatury všech metod předka.
- činí to práci s modelem přehlednějším: mnohdy totiž při modelování pracujeme s předkem namísto s konkrétními potomky (můžeme to udělat všude tam, kde lze použít jakéhokoliv potomka ze stromu inheritancí) a hned v bázové třídě vidíme, jaké chování zajišťují všechny její potomky (nemusíme tedy neustále znovu a znovu prozkoumávat celý strom inheritancí : společné rysy máme *po ruce* v bázové třídě).

### 1.3.4 ABSTRAKTNÍ METODY A TŘÍDY

**Metodě, jejíž implementace v předkovi úplně chybí, říkáme abstraktní metoda. Třídě, která má alespoň jednu abstraktní metodu, říkáme abstraktní třída.**

V předchozím příkladu jsme si ukázali, že metody předka spočítejObsah() a nakresli() musí být v potomcích předefinovány. Pokud všichni potomci překrývají metodu *po svém*, tak ani nemá smysl v předkovi tuto metodu implementovat - chceme ji ale přesto v předkovi uvést (můžou nás k tomu vést například důvody uvedené na konci předchozího odstavce). Takovýmto metodám bez implementace říkáme **abstraktní metody**. Ta třída, která má alespoň jednu abstraktní metodu, není nadefinována do té míry, abychom mohli vytvořit instanci této třídy (abstraktní metoda této třídy je prázdná). Takovéto třídě říkáme **abstraktní třída.** Metodě, která není abstraktní, pak říkáme **konkrétní metoda**, třídě, která není abstraktní, pak říkáme **konkrétní třída**. Předchozí obrázek tedy můžeme zakreslit (s využitím notace UML, kde se abstraktní metody a třídy označují *kurzívou*) takto:

### 1.3.5 POLYMORFISMUS

Kolem polymorfizmu se dělá někdy snad až příliš mnoho humbuku, ale pokud *přemýšlíme objektově*, nic zvláštního na tom není.

**Polymorfní metody: takové metody, které se vyskytují u více objektů (tj. mají u různých objektů stejnou signaturu), ale mají odlišnou implementaci.**

**Příklad:** Máme metody spočítejObsah() a nakresli(). V konkrétních třídách mají tyto metody stejnou signaturu, avšak v každé třídě je jiná implementace: metoda nakresli() spuštěná v objektu třídy **kruh** bude kreslit kruh, metoda nakresli() spuštěná v objektu třídy **pravoúhelník** bude kreslit čtverec nebo obdélník. Tyto metody jsou tedy polymorfní - v různých třídách mají různé chování.

## SHRNUTÍ

V této kapitole je obsažen úvod do objektově orientovaného modelování. Jsou zde stručně shrnuty důvody přechodu od strukturovaného pojetí k objektovému. Jsou zde vymezeny a ozřejměny základní pojmy, které budou využívány v dalším výkladu.

## LITERATURA

Tento text vznikl za využití této literatury:

http://mpavus.wz.cz
http://uml.czweb.org/index.htm

**[2] [Arl2003]** ARLOW, J. NEUSTADT, I. UML a unifikovaný proces vývoje aplikací: Objektově orientovaná analýza a návrh prakticky. Přel. Bogdan Kizska. Brno: Computer Press, 2008. 387 s. ISBN 80-7226-947-X.

**[8] [Kan2004]** KANISOVÁ, H. Miller, M. UML srozumitelně, 1.vyd, Brno: Computer Press, 2004. 158 s. ISBN 80-251-0231-9.

**[9] [Lar2001]** LARMAN, C. Applying UML and patterns : an introduction to object-oriented analysis and design and the unified process. 2.vyd. Upper Saddle River: Prentice Hall PTR,

# 2 PRINCIP A ZÁKLADNÍ ELEMENTY JAZYKA UML

Modelovací jazyk UML je souhrnem grafických notací k vyjádření analytických a návrhových modelů. UML je jazyk, který dává předpoklady a možnosti pro modelování jednoduchých i složitých aplikací s využitím stejné formální syntaxe. Výsledky své práce tak mohou sdílet všichni návrháři podílející se na vývoji systému. Vybrané segmenty jazyka UML jsou pochopitelné i pro zadavatele aplikace a dávají možnost objasnění požadavků uživatelů na vytvářený systém. Diagramy UML se soustředí vždy na právě jeden pohled na vyvíjený systém. Jeden druh diagramu nezachycuje navrhovaný systém jako celek. Ten je zachycen komplexem jednotlivých diagramů UML.

UML je rovněž jazyk pro vizualizaci, specifikaci, stavbu a dokumentaci software zajišťující chod informačních systémů. Ve světě IT existují různé metodické postupy, které vycházejí z modelovacích technik UML a rozšiřují je o vlastní doporučené postupy, další diagramy a techniky jako je specifikace požadavků a procesní modelování. Nejznámějšími metodikami jsou RUP (Rational unified process) firmy Rational (nyní ve vlastnictví IBM) a Select Business Solution, viz např. UML srozumitelně, [Kan004].

## UML = unified modeling language (tj. *unifikovaný modelovací jazyk*)

**U**nifikovaný: unifikuje Booch, OMT a Objectory modelovací jazyky

**M**odelovací: UML je jazyk pro **specifikaci, visualizaci, konstrukci a dokumentaci** artefaktů SW systémů.

UML je však využitelný i pro business modelování i pro modelování ne-SW systémů. V UML lze modelovat jakýkoliv typ aplikace běžící na jakémkoliv typu a kombinaci HW, OS, programovacím jazyku a sítě. Lze modelovat distribuované aplikace. Pro UML je nejpřirozenější modelování pro OO jazyky a prostředí (jako jsou např. C++, Java, C#), ale lze modelovat i pro neobjektové jazyky (např. Fortran, VB, COBOL). **UML profiles** nám pomůže v modelování transakčních, real-time a fault-tolerant systémů.

**L**anguage (jazyk): UML není **programovací** jazyk, ale je to jazyk, neboť má:

- **syntaxi** (grafickou), tj. pravidla, dle kterých jsou elementy jazyka sestavovány do výrazů
- **sémantiku**, tj. pravidla, dle kterých je syntaktickým výrazům přiřazen význam

UML samo v sobě obasahuje mechanismy pro jeho rozšíření (dle potřeb konkrétního projektu nebo konkrétního vývojáře)

Další vlastnosti UML :

- otevřený standard
- podporuje celý vývojový cyklus
- podporuje různě aplikační oblasti
- založeno na zkušnostech potřebách komunity uživatelů
- podporováno celou řadou nástrojů
- přispěvatelé: Booch (Booch metoda), Rumbaugh (OMT), Jacobson (OOSE), Shlaer-Mellor (object lifecyles), Odell (clasification), Wirfs-Brock (Responsibilities), Embley (Singleton classes and high level view), HP Fusion (operation descriptions and message numbering), Gamma a kolektiv (frameworks and patterns), Harel (statecharts), Meyer (before and after conditions)

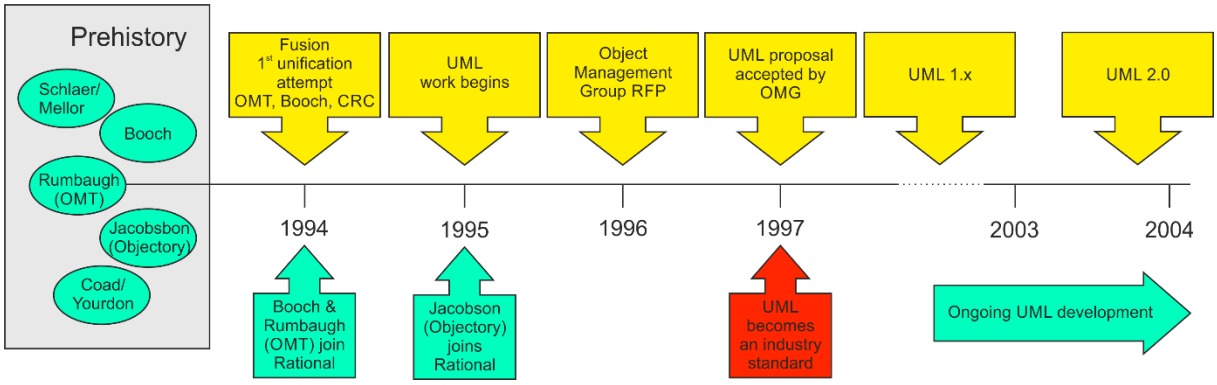
## 2.1 HISTORIE :

1966 první objektově orientovaný jazyk Simula (O. J. Dahl, K. Nygaard)
1967 Simula67 : třídy, dědičnost, polymorfizmus
70. léta první **čistě** objektový programovací jazyk Smalltalk : pojem **objekt** (Xerox)
80. léta éra *smíšených jazyků* Turbo Pascal, C++ : umožňovaly programovat objektově, ale tento přístup nevyžadovaly
přelom 80. a 90. let řada metod pro OOA a OOD + CASE na nich založené:

- **CRC** karty : class-responsibility-collaboration (z prostředí Smalltalk, Xerox)
- **Booch**, kniha : Object-Oriented Analysis and Design with Applications, 2nd edition, Addison-Wesley, 1994 (Grady Booch, Rational)
- **OMT**, kniha : Object-Oriented Modeling and Design, Prentice Hall, 1991 (J. Rumbaugh (General Electric) a další)
- **případ užití** (OOSE), kniha : Object-Oriented Softvare Enginering : A Use Case Driven Approach, Addison-Wesley, 1992 (I. Jacobson a další)

1994 J. Rumbaugh se přidává k G. Boochovi a firmě Rational Software
1995 Unified Method verze 0.8 (G. Booch, J. Rumbaugh)
1995 I. Jacobson (Objectory) se přidává k G. Boochovi a J. Rumbaughovi (pokud se někde dočtete o *The Three Amigos* či *Los Tres Buddies*, tak vězte, že je to přezdívka tria Booch+Rumbaugh, Jacobson :-)
1996 UML 0.9, 0.91 do prací na UML se zapojují např.: Digital Equiment, HP, i-Logix, IBM, MCI, Microsoft, Oracle, TI, Unisys
1997 rational : 1.0, 1.1 OMG "adoptovalo" UML UML 1.0 (OMG) = UML 1.1 : standard, většinou nazývaný 1.1
1998 UML 1.2 : interní verze, nepodstatné úpravy
1999 UML 1.3 : změny v use cases (namísto <<uses>> a <<extends>> je zavedeno <<include>>, <<extend>> a generalizace) a activity diagramech
2001 UML 1.4
2001-2005 práce na UML 2.0
2005 : UML 2.0 Části UML 2.0 specifikace: Superstructure, Infrastructure, Object Facility (MOF), Object constrain language (OCL), popis XML interakcí mezi jednotlivými modely

**Obrázek 2:** Historie vzniku UML




Zdroj: http://mpavus.wz.cz

## 2.2 UML - SKLADBA

Skladba definuje z jakých *kamenů* se UML skládá.

Tyto kameny jsou pouhopouhé tři:

- **předměty (things)**, tj. elementy modelu definující:
  - **strukturní abstrakce (structural things)** : podstatná jména (modelu UML), tj. elementy: třída (class), rozhraní (interface), spolupráce (collaboration), případ užití (use-case), aktivní třída (active class), komponenta (component), uzel (node)
  - **chování (behavioral things)** : slovesa (modelu UML), tj.: interakce (vztahují se ke komunikaci mezi objekty - např. zprávy (messages), spoje (links)) a stavový stroj (specifikuje sekvenci stavů objektu pomocí stavů, přechodů, událostí a aktivit)
  - **seskupení (grouping things)** : balíčky (packages) seskupující (dle potřeby) prvky modelu
  - **poznámky (anotational things)** : poznámky s přidanými informacemi
- **relace (relationships)**, tj. elementy modelu spojující spolu dva (či více) předmětů - **strukturních abstrakcí** (také mohou spojovat **seskupení**); jejich typy jsou:
  - **závislost (dependency)** : znázornění vztahu, kdy změnou v jednom elementu je ovlivněn jiný (závislý) element
  - **asociace (association)** : spojení definující vztah mezi elementy
  - **zobecnění (generalization)** : vztahy generalizace-specializace, tj. jeden element je specializací jiného elementu
  - **realizace (realization)** : jeden element je realizací jiného elementu
- **diagramy**: model je souhrn všech předmětů a relací, jejichž znázornění v jednom obrázku by bylo nepřehledné, mnohdy zcela nemožné; naproti tomu diagram je jeden *průhled*, *okénko*, kterým se díváme na model; diagramů je v UML několik typů, ale většinou máme v jednom projektu i vícero diagramů jenoho typu (ať už z důvodu velikosti modelu, nebo chceme v různých diagramech zobrazit pohled na systém z různých *úhlů*, či se v různých diagramech chceme zaměřit na různé aspekty modelovaného systému). Máme tedy devět diagramů ve dvou skupinách:
  - statický model (zaměřený na systémovou strukturu) :
    - diagram tříd
    - diagram komponent
    - diagram nasazení
  - dynamický model (zaměřený na chování systému) :
    - diagram případu použití
    - sekvenční diagram
    - diagram spolupráce
    - stavový diagram
    - diagram aktivit
    - objektový diagram

## 2.3 UML - MECHANISMY

UML má čtyři mechanismy, které se prolínají celým jazykem.

Tyto mechanismy jsou:

- **specifikace**: každý element může (či měl by) být specifikován textem, který popisuje sémantiku tohoto elementu. Tato specifikace upřesňuje, blíže popisuje, udává smysl modelovaného elementu. Popisuje *business* pravidla elementů (tudíž má největší význam u elementů popisujících problémovou doménu).
- **ozdoby (adornments)** : další informace známé o elementu modelu. Každý element může být vyjádřen jednoduchým tvarem, ale je možno k němu přidávat i další informace - ozdoby. Proč je těchto ozdob u elementu zobrazeno někdy více a někdy méně:
  - model vytváříme postupně: zpočátku máme málo informací, které postupně doplňujeme
  - tvorbou určitého diagramu sledujeme určité cíle - nechceme v něm zobrazit ty podrobnosti, které jsou v tomto nepodstatné z hlediska toho, co právě chceme zdůraznit (jedno z prvořadých hledisek při tvorbě diagramu je totiž snadná čitelnost)
- **podskupiny (common division)** : udávají, jak je možno rozdělovat (seskupovat) jednotlivé elementy; první způsob dělení:
  - **klasifikátor a instance**: pro dva elementy UML jsme si toto rozdělení již probrali - objekt je **instance**, kdežto třída je **klasifikátor**. Podobný vztah klasifikátor-instance lze nalézt pro další elementy UML. Každý element je buď klasifikátor a neboinstance, a toto rozlišení je velmi důležité. Osvojením tohoto dělení si usnadníte komunikace mezi členy týmu (stačí říct: .... klasifikátor elementu xxxx ...., .... instance elementu yyyyy ..... a bez dalšího vysvětlování je jasné, o co jde), můžeme se s tímto setkat i v CASE nástrojích a v literatuře.
  - **rozhraní a implementace**: zalistujme výše v tomto kurzu do kapitoly popisující objekt - mluvili jsme o **protokolu zpráv**, což je rozhraní objektu. Implementace pak jsou metody, které *řeší*, implementují, toto rozhraní. (Obecně však může být definované rozhraní pro každý klasifikátor). Toto důrazné oddělení má dvojí praktický význam:
    - k tomu, abychom použili (již hotový) objekt, nemusíme znát jeho implementaci - stačí nám znát jeho rozhraní; programátoři (i *neobjektoví*) vlastně tohoto využívají při volání knihovních funkcí: programátor v jazyce C nemusí znát, jak je vnitřně vyřešena funkce **printf**, ale zná její rozhraní, tedy ji může používat
    - a z druhé strany: *vnitřek* objektu může být (v budoucnu) libovolně změněn - ale jen tak, aby jeho klienty (tj. ty, které využívají jeho služby) nemuselo být nutno revidovat - jinak řečeno: i po úpravách musí objekt správně implementovat dohodnuté rozhraní; tedy ten, kdo vytváří/mění *vnitřek* objektu, nemusí nic vědět o tom, jak a kým bude objekt použit - stačí, když bude správně implementovat rozhraní
- **mechanismy rozšiřitelnosti**: jakyk UML sám v sobě obsahuje připravené mechanismy umožňující rozšířit jazkyk tak, aby vyhovoval momentálním potřebám. Máme k dispozici tři mechanismy rozšiřitelnosti:
  - **omezení (constraints)** : jde o text ve složených závorkách {}. Podmínka či pravidlo v tomto textu musí být vždy splněna

- **stereotypy (stereotypes)** : s jejich pomocí lze z existujího elementu vytvořit nový. Vytvoříme ho tak, že název stereotypu vložíme do dvojitých ostrých závorek: <<novy_stereotyp>>. Stereotyp může mít rovněž přiřazen nový symbol - časté využití bývá např. v diagramu nasazení pro vytvoření symbolů tiskáren, serverů, notebooků apod. Některé stereotypy jsou již součástí jazyka UML a ještě se s nimi v tomto kurzu setkáme.
- **označené hodnoty (taggedd values)**: umožňuje přidávat nové vlastnosti k elementům modelu. Zavedeme ji přidáním názvu s připojenou vlastní hodnotou ve složených závorkách, např. {autor=pavus, verze=0.1}

## LITERATURA

Tato kapitola zavádí základní pojmy jazyka UML. Text vznikl využitím literatury:

http://mpavus.wz.cz
http://uml.czweb.org/index.html

**[1] [Amb2004]** AMBLER, S. W. The Object Primer: Agile Model-Driven Development with UML 2.0. 3. edition. Cambridge University Press, 2004. 572 s. ISBN 0521540186.

**[4] [Boo1998]** BOOCH, G. JACOBSON, I. RUMBAUGH, J. The Unified Modeling Language User Guide. 1.vyd. Addison Wesley, 1998. ISBN 0- 201-57168-4.

**[8] [Kan2004]** KANISOVÁ, H. Miller, M. UML srozumitelně, 1.vyd, Brno: Computer Press, 2004. 158 s. ISBN 80-251-0231-9.

# 3 POPIS JAZYKA UML – PŘÍPADY UŽITÍ

Případy užití, typové úlohy nebo chcete-li užitné případy. Se všemi těmito překlady originálu **Use Case** se lze v odborné literatuře setkat. V následujícím textu se přikloníme k terminologii „případ užití, který odpovídá anglickému originálu.

Již dlouhou dobu se vývojáři bez ohledu na svou orientaci na vývojové prostředí snaží modelovat typické interakce uživatelů se systémy, aby tak lépe pochopili skutečné požadavky uživatelů na budoucí systém a zároveň aby vymezili rozsah navrhované aplikace.

Případy užití zachycují přesně funkčnost, která bude budoucím informačním systémem pokryta a vymezují tak jednoznačně rozsah prací. Každý případ popisuje jeden ze způsobů systému, popisuje tedy jednu jeho požadovanou funkčnost. Pro co nejnázornější výklad pojmu USE CASE a pro výklad UML diagramů v dalších kapitolách si popíšeme konkrétní modelový případ návrhu informačního systému v opravárenské firmě. Vytvoříme tzv. případovou studii.

## 3.1 PŘÍPADOVÁ STUDIE¨

Celým textem této učební pomůcky se bude prolínat případová studie převzata z publikace [Kan,2004]. V této kapitole si objasníme problémovou oblast (setkáte se také s pojmem problémová doména).

Ukázky z případové studie uvedené v jednotlivých kapitolách nelze pokládat za kompletní studii, která by vyčerpávajícím způsobem seznámila s teorií UML a jejím použitím na konkrétním projektu. Cílem bylo demonstrovat praktické použití UML na konkrétním projektu s cílem zjednodušit a zprůhlednit ukázky se v této studii uvažuje pouze o zavádění nových dat a jejich aktualizaci. Tato studie neřeší účetní stránku modelované problematiky pro složitost, která by způsobila nepřehlednost analýzy. Z tohoto důvodu byla ponechána stranou i důležitá část informačního systému a to odstraňování a archivace dat.

Ukázky by měly posloužit jako příklad metod UML pro začínající uživatele a tomu je přizpůsobena složitost studie a prezentovaných ukázek.

V průběhu studia této učební pomůcky narazíte na demonstrované obrázky modelů UML. Co nejvíce modelů pochází právě z této modelové studie. Ne vždy bylo tento princip možno dodržet. Například nelze v jednom modelu najít příklady na všechny typy vazeb mezi třídami. Na některých místech se proto setkáte s ukázkami, které nepochází z modelové studie. Cílem není dokonalá analýza dané problematiky, ale demonstrace technik UML. V dalším textu se setkáte s příklady jednotlivých modelovacích technik, které ho budou provázet celím výkladem.

Studie vychází z modelové situace, kdy softwarová firma získala zakázku na analýzu, návrh a vývoj informačního systému, který by funkčně pokrýval potřeby sběrny oprav elektrospotřebičů. Představme si situaci, kdy existuje sběrna oprav domácích elektrospotřebičů (dále jen „zákazník“) a předmětem podnikání je zprostředkování oprav elektrospotřebičů ve značkových i neznačkových smluvních servisech podle druhů. Pro jednoduchost budeme v popisované modelové studií uvažovat pouze o jednom modulu informačního systému, a to modulu pro zprostředkování oprav. Záležitosti ekonomické, tj. účetnictví, fakturace atd. nejsou předmětem případové studie.

Pracovníci softwarové firmy absolvovali několik úvodních jednání se zástupci zákazníka. Výsledky těchto jednání byly sumarizovány do podoby uživatelských požadavků a představy o fungování budoucí aplikace.

Zákazník uplatnil svou představu, že bude na vlastním HW provozovat aplikaci na evidenci oprav elektrospotřebičů, do které budou mít přístup i jednotlivé smluvní servisy. Servisy budou mít zároveň za povinnost v aplikaci aktualizovat stav oprav, aby sběrna měla vždy přehled o stavu prací.

Pro pochopení souvislostí uvedeme nyní seznam požadavků zákazníka:

1. Příjem zakázky na opravu spotřebiče
2. Výdej zakázky majiteli
3. Správa číselníků
4. Monitoring oprav
5. Evidence zákazníků
6. Vyhodnocení oprav
7. Vyřízení reklamace opravy
8. Oprava spotřebiče v servisu

### *3.1.1 PŘÍJEM ZAKÁZKY NA OPRAVU SPOTŘEBIČE*

Podpora příjmu zakázky na opravu spotřebiče.

Zákazník přichází do obchodu a chce vyřídit opravu, podle uplynulé doby od nákupu jde o opravu záruční nebo pozáruční. Pracovník prodejny přijme spotřebič do opravy, vystaví zakázkový list se slovním popisem závady. U záručních oprav převezme záruční list a seznámí zákazníka s odhadovanou cenou (u záruky = 0) a orientační délkou opravy.

Zakázkový list obsahuje:

- číslo dokladu (generováno automaticky jako číselná řada v rámci roku)
- údaje o zákazníkovi (jméno, příjmení, adresa, telefon domů a do zaměstnání, mobil, e-mailová adresa, OP).

Při zadávání údajů o zákazníkovi pracovník sběrny ověří, zda se jedná o první zakázku zákazníka, vyhledá a zaktualizuje údaje o zákazníkovi.

Údaje o opravovaném předmětu druh (z číselníku, např. televizor, pračka, video, ...), značka a typ (např. Philips 24VT, Sony 556 apod.), výrobní nebo další údaje pro identifikaci (např. barva), popis závady, datum předpokládaného vyřízení, předpokládaná cena opravy.

Pracovník odešle zboží do příslušné servisní opravny dle druhu spotřebiče (Sony, Philips, ....) kurýrní službou. Zároveň poznačí v aplikaci datum odeslání do opravy a potvrdí odeslání. Systém automaticky vygeneruje upozornění v podobě mailu pro servis, že jim byla odeslána zakázka s odkazem na zakázkový list. Aplikace zajistí tisk průvodky k dopravě.

### *3.1.2 VYDEJ ZAKÁZKY MAJITELI:*

Podpora pro předání opraveného spotřebiče zákazníkovi, vyúčtování ceny, vyřízení dokladů.

Z opravny se vrací opravení spotřebič spolu s dokladem o servisním zásahu, na kterém je uvedeno, kdo opravu provedl, jaký náhradní díly byly při opravě použity, včetně jejich

ceny, jaké pracovní úkony byly provedeny a jejich cena. Sběrna připočte k ceně opravy svou přirážku, která je volitelná u každé zakázky, standardně 5% z ceny opravy.

Po obdržení spotřebiče ze servisu pracovník kontaktuje dohodnutím způsobem (telefon, mail) zákazníka k vyzvednutí zakázky z opravy.

Zákazník přichází, zaplatí opravu a odnáší si opravený spotřebič. Aplikace musí podporovat proces odbavení zákazníka.

### *3.1.3 SPRÁVA ČÍSELNÍKŮ APLIKACE*

Požadavek řeší správu těchto číselníků:

1. Výrobci (např. Sony, Philips, …)
2. Typy (televize, video, …)
3. Servisy (smluvní servisy sběrny)
4. Státy (výrobců)

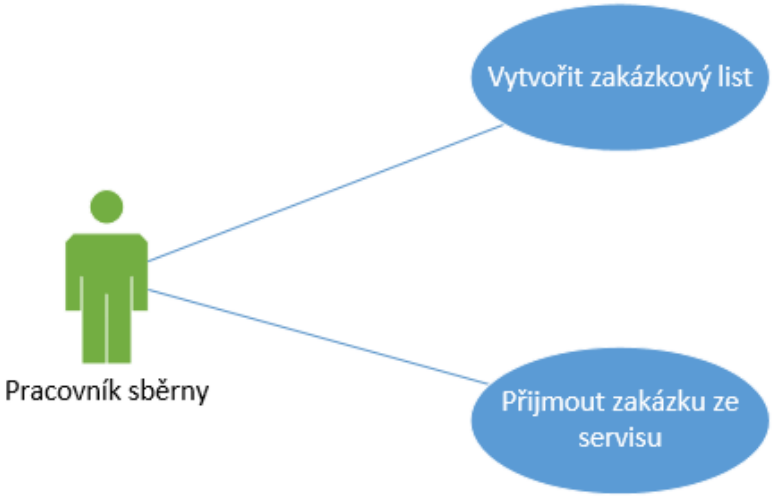
Obdobně se popíší všechny požadované funkce informačního systému.

## 3.2 AKTÉŘI

V rámci firemních procesů vystupuje řada firemních aktérů. Ne každý z nich ale komunikuje s daným informačním systémem. Pod pojmem aktér budeme rozumět roli, ve které vystupuje uživatel v rámci jeho komunikace se systémem. Na obrázku 2 je znázorněn velmi jednoduchý diagram případů, které budou podrobněji vysvětleny dále.

Diagram zobrazuje jen výsek z případové studie, potřebný pro vysvětlení aktérů. Na tomto obrázku je znázorněn symbolem postavičky pouze jeden aktér *Pracovník směny*. V naší případové studii Sběrna oprav je zaměstnána celá řada pracovníků, ale v informačním systému vystupují všichni ve stejné roli.

**Obrázek 3:** Aktér = uživatelská role vůči systému



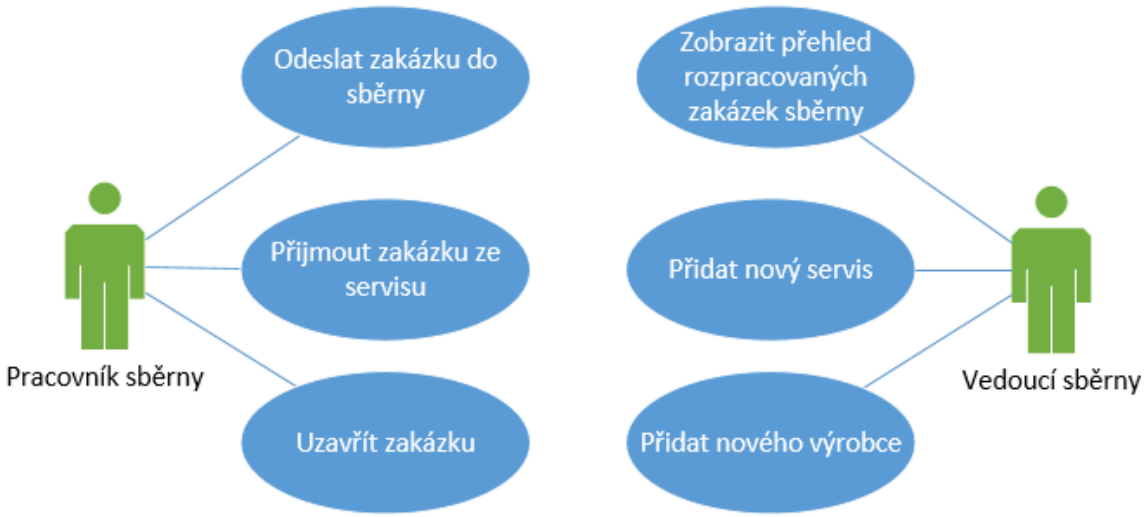

Zdroj: **[Kan2004]**

Jeden fyzický uživatel může vůči systému vystupovat ve více rolích. Na obrázku 3 je znázorněna situace se dvěma aktéry, přičemž v rolích *Pracovník směny* a *Vedoucí směny* může vystupovat jeden fyzický pracovník.

Aktéři provádí případy užití. V systému může jeden aktér provádět řadu případů užití a obráceně, jeden případ užití může být prováděn více aktéry.

Pohled na případy užití očima aktérů nám může výrazně usnadnit identifikování případů užití v rozsáhlých systémech. Tam bývá zpravidla obtížné identifikovat kompletní seznam případů užití, a proto je lépe nejprve definovat aktéry a potom pro každého z nich rozpracovat případy, které vykonává.

**Obrázek 4:** Uživatel ve více rolích vůči systému




Zdroj: **[Kan2004]**

Ačkoli pro znázornění aktérů zavádí UML Symbol postavičky, nemusí jimi vždy být lidé. Jako aktér může stejně vystupovat externí systém, který potřebuje informace z našeho systému, ale dokonce i čas, např. v situacích, kdy se v definovanou hodinu spouští v systému bez zásahu obsluhy pravidelné zpracování (denní apod.).

## 3.3 PŘÍPADY UŽITÍ

Prvním, kdo zviditelnil případy užití, byl Ivar Jacobson v roce 1992. Okamžitě po vydání jeho knihy byly případy užití přijaty komunitou objektově orientovaných vývojářů jako základní element při plánování, vývoji a realizaci projektu. Abychom dovedli odpovědět na otázku „Co vlastně znamená případ užití“ musíme nejprve vysvětlit pojem **scénář případu užití**. Scénář je sekvence kroků popisujících interakci mezi aktérem a systémem. Takže například v systému sběrny oprav elektrospotřebičů (viz případová studie, můžeme vytvořit scénář, viz obrázek 5:

**Obrázek 5:** Příklad scénáře případu užití

| Krok | Role | Akce |
|---|---|---|
| 1 | Uživatel | spustí volbu *Založit zakázku* |

| | | |
|---|---|---|
| 2 | Systém | zobrazí formulář detailu zakázky a zpřístupní údaje pro pořízení |
| 3 | Uživatel | pořídí vstupní informace zakázky, jedná se o tyto údaje … |
| 4 | Uživatel | aktivuje výběr zákazníka z evidence sběrny |
| 5 | Systém | zobrazí formulář seznamu zákazníků v abecedním třídění podle příjmení a jména, přičemž se zobrazují údaje příjmení, jméno, místo, ulice |
| 6 | Uživatel | vybere zákazníka ze zobrazeného seznamu a přiřadí ho k zakázce |
| 7 | Systém | zavře formulář seznamu zákazníků a vrátí se do editace zakázky |
| 8 | Uživatel | aktivuje zobrazení seznamu spotřebičů |
| 9 | Systém | zobrazí formulář seznamu spotřebičů v třídění podle názvu spotřebiče |
| 10 | Uživatel | vybere spotřebič ze seznamu |
| 11 | Systém | deaktivuje formulář seznamu spotřebičů a vrátí se k editaci zakázkového listu |
| 12 | Uživatel | zapíše údaje o poruše spotřebiče a dá pokyn k tisku zakázkového listu |
| 13 | Systém | vytiskne zakázkový list a uzavře formulář zakázky |

Zdroj: **[Kan2004]**

Všimněte si ne náhodné podobnosti takto zapsaného scénáře případu užití se skutečnými scénáři, např. divadelních her nebo filmů.

Dostáváme se k definici případu užití. **Případ užití je sada scénářů, které spojuje dohromady společný cíl.**

Další, alternativní cesty v případech užití představují další scénáře. V praxi bývá téměř pravidlem, že případ užití má základní scénář typu „všechno šlo hladce“ a řadu alternativních scénářů, které představují postup při zjištění různých chyb, mimořádných stavů a z nich odvozených větví průchodu. K hlavnímu scénáři na obr. 4 tak můžeme doplnit alternativní scénář podle obr. 5.

**Obrázek 6:** Případ užití – Založení nového zákazníka

| Krok | Role | Akce |
|---|---|---|
| 4a1 | Uživatel | zvolí nabídku *Nový zákazník* |
| 4a2 | Systém | zobrazí prázdný formulář zákazníka se zpřístupněnými údaji |

| | | |
|---|---|---|
| | | zákazníka |
| 4a3 | Uživatel | pořídí údaje zákazníka: příjmení, jméno, titul, adresa, telefon |
| 4a4 | Systém | založí nového zákazníka a převezme jeho identifikaci do zakázkového listu |

Zdroj: **[Kan2004]**

## 3.4 POPIS PŘÍPADU UŽITÍ

Jak konkrétně má vypadat popis případu užití? Na tuto otázku lze nalézt řadu odpovědí. Samotné UML v tomto ohledu nedefinuje žádný standard, a tak zůstaňme u doporučení osvědčených v praxi:

- Název případu užití by měl být tvořen pomocí slovesné vazby (představuje nějakou akci v systému, tedy např. *Přijmout spotřebič do opravy* místo *Příjem spotřebiče do opravy*).
- Pro zápis hlavního úspěšného scénáře použijte jednoduchou sekvenci číslovaných kroků, viz obr. 4.4.
- Pro zápis případných rozšíření v podobě alternativních scénářů použijte opět číslovanou sekvenci odkazující se na hlavní sekvenci, viz obr. 4.5.
- Používejte stručné a srozumitelné věty.
- V rámci týmu přijměte a dodržujte pravidla pro zápis větvení ve scénářích případů užití, tato pravidla by měla být kompromisem mezi zápisem nového případu užití pro každou větev samostatně na straně jedné a jediným komplexním případem užití na straně druhé.
- Součástí pravidel mohou být také sjednané konstrukce v podobě jakéhosi metajazyka, které stručně a srozumitelně vyjadřují větvení podmínek, jako např. Když – potom – jinak, případně vyjadřují opakování skupiny činností ve scénáři, dokud je splněna nějaká podmínka, např. PRO podmínka …, použití těchto konstruktů ilustruje obr. 4.6.
    - Vyjadřuje se za pomoci slovníku aktéra (řešené problémové oblasti) a to jednotně jak v rámci jednotlivých případů užití, tak v rámci týmu (např. termín „servis“ by se měl ve všech případech užití vyskytovat jako „servis“ a ne někde jako „opravna“ apod.) (za tímto účelem se doporučuje v rámci týmu zřídit jednotný slovník používaných výrazů)

**Obrázek 7:** Případ užití: Příjem spotřebiče do opravy

| Krok | Role | Akce |
|---|---|---|
| 1 | Obsluha | dá pokyn k založení zakázkové listu |
| 2 | Systém | zobrazí formulář zakázkového listu se zpřístupněnými údaji |
| 3 | Obsluha | aktivuje výběr zákazníka ze seznamu existujících |

| | | |
|---|---|---|
| 4 | Systém | zobrazí seznam existujících zákazníků sběrny |
| 5 | Uživatel | POKUD zákazník dosud neexistuje v evidenci sběrny,<br>pořídí údaje zákazníka …<br><br>JINAK vybere zákazníka z evidence<br>KONEC – POKUD |
| 6 | Systém | předá údaje zákazníka do zakázkového listu |
| 7 | … | … |

Zdroj: **[Kan2004]**

## 3.5 VSTUPNÍ A VÝSTUPNÍ PODMÍNKY

Kromě uvedeného popisu případu užití formou základního eventuálně alternativních scénářů je možné popis dále upřesnit pomocí přídavných sekcí. Takovými sekcemi mohou být **vstupní podmínky případu užití** (Pre-Conditions), definující předpoklady, které musí být splněny, aby případ užití mohl být zahájen, nebo naopak **výstupní podmínky** (Post-Conditions), určující kritéria, která musí být splněna po skončení případu užití. Jinými slovy, vstupní podmínky představují omezení stavu systému před spuštěním případu užití, zatímco následné podmínky omezující stav systému po skončení případu užití. Doplňme, že popis vstupních a výstupních podmínek by měl být stručný, jasný a vždy snadno vyhodnotitelný jako pravda/nepravda.

Vstupní podmínky je potřeba definovat tehdy, pokud případ užití nemůže být spuštěn bez jejich naplnění. Výstupní podmínky je ve většině případů přípustné jednoduše vynechat, pokud výsledek případu užití je zřejmý nebo pokud nedochází k významné změně stavu systému. Svůj význam mají tehdy, pokud případ užití znamená dosažení nějakého stavu důležitého pro aktéry případu užití. Chcete-li se rozhodnout, zda v případu užití uvést výstupní podmínky, pokuste se odpovědět na tyto otázky:

- Přivede dokončení daného případu užití systém do specifického stavu, který je zároveň vstupní podmínkou jiných případů užití? Pokud ano, zapište to jako výstupní podmínku …
- Jsou možné výsledky daného případu užití zřejmé tak, že jim vývojáři, testeři a uživatelé rozumí? Pokud ne, zapište výstupní podmínky.

Jako příklad výstupní podmínky k případu užití na obr. 4.4 uveďme:

- Stav zakázky je nastaven na hodnotu *„převzato do opravny“*.

To je ukázka důležitého stavu zakázky, která je zároveň vstupní podmínkou pro případ užití *Odeslat spotřebič do servisu.*

V případě, že u případu užití zavedete více vstupních nebo výstupních podmínek, je třeba dát si pozor na jejich vyhodnocování. Mezi vstupními podmínkami platí operátor AND

(musí být splněny všechny zároveň), zatímco u výstupních podmínek platí operátor OR (musí být splněna alespoň jedna z nich).

## 3.6 POSTUP TVORBY POPISU PŘÍPADU UŽITÍ

Různých doplnění či upřesnění popisu případů užití uvedených v předchozích odstavcích může být více. Kolik, to závisí na závažnosti problematiky, kterou případ užití popisuje a míře rizika z toho plynoucí. Čím vyšší riziko, tím vyšší potřeba upřesnění či doplnění popisu případu užití.

Pro rozpracování případů užití můžeme doporučit, abyste si uvědomili, že tvorba případů užití je iterativním procesem, který začíná nejprve jen názvy případů užití, v dalších iterací doplníte nejprve stručné popisy a konečně detailní specifikace. V prvních iteracích se zaměřte jen na nejobvyklejší scénář. Ten totiž přináší největší užitek z hlediska pochopení případu užití. Teprve v dalších iteracích rozpracujte případné alternativy (zpravidla se jedná o ošetření chyb uživatele) a podle jejich povahy a množství založte alternativní scénáře již existujících případů užití nebo alternativní případy užití.

Do jakých detailů mají popisy případů užití jít? Obecně lze konstatovat, že míra detailnosti by měla být taková, aby budoucí uživatelé systému byli schopni popis případu užití odsouhlasit a aby byli přesvědčeni, že budoucí systém bude mít tu správnou funkčnost, kterou požadují. Platí totiž pravidlo, že systém bude poskytovat právě tolik funkčnosti, kolik je popsáno formou případů užití, ne víc a ne míň.

Během rozpracování případů užití se doporučuje rozdělit takové případy užití, které se stávají příliš komplikovanými. V průběhu konstrukce může také dojít k rozdělení případů užití podle toho, jak budou realizovány v jednotlivých iteracích – přírůstcích. Vyřešte vždy základní případ a teprve dodatečně jeho varianty.

### SHRNUTÍ

V této kapitole je uveden případ užití, který modeluje interakce uživatelů s vytvářeným informačním systémem. Případ užití je převzat z literatury jako charakteristický příklad.

### LITERATURA

Použitý zdroj:

**[Kan2004]** KANISOVÁ, H. Miller, M. UML srozumitelně, 1.vyd, Brno: Computer Press, 2004. 158 s. ISBN 80-251-0231-9.

# 4 POPIS JAZYKA UML – MODELOVÁNÍ TŘÍD A OBJEKTŮ

## 4.1 ZÁKLADNÍ POPIS DIAGRAMU TŘÍD

Diagram tříd (Class Diagram) představuje „statický pohled na modelovaný systém“ [Buch2007] a jeho úkolem je znázornit typy objektů v systému a jejich vztahy. [Fow2003] Návrh tříd, jejich odpovědností a následné vytvoření tohoto diagramu je jedním z prvních a základních kroků analýzy navrhovaného programového systému. [Buch2007] Díky tomu, že diagram tříd zachycuje pravidla modelovaného systému, je nejdůležitějším podkladem jak pro forward engineering, tak pro reverse engineering. [Pen2003]

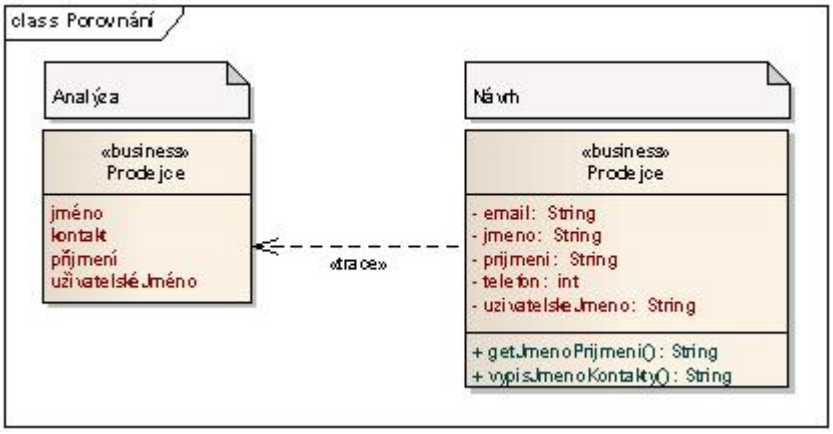
Při tvorbě diagramu tříd je nutné vzít v úvahu jeho účel a rozlišit, zda potřebujeme vyjádřit požadavky na modelovaný software nebo získat podrobný popis designu atd. Z tohoto důvodu se dle [Buch2007] rozeznávají tři úrovně modelu tříd – konceptuální, designová a implementační.

Konceptuální (doménový, analytický) model (viz. Obrázek 2) je vytvářen za účelem analýzy požadavků na software. Obsahuje pouze tzv. byznys třídy (business classes), které modelují problémovou oblast a jsou tedy součástí slovníku problémové domény (slovníčku pojmů). [Buch2007] U jednotlivých tříd se uvádí obvykle jen názvy klíčových atributů a některých klíčových metod. Pokud je diagram vytvářen pouze za účelem znázornění relací mezi třídami, je možné atributy i metody vynechat. [Arl2008]

Designový model (model návrhu) vychází z modelu konceptuálního, který rozšiřuje a zpřesňuje například o viditelnosti atributů a metod, datové typy apod. Dále do modelu přidává třídy uživatelského rozhraní (presentation classes) a třídy obsluhující systémové události (control classes). [Buch2007] Z jedné třídy v analytickém modelu se tedy může stát v designovém modelu více návrhových tříd. Mezi analytickými třídami a třídami návrhu existuje relace typu trace(viz obrázek 1).

[Arl2008] Implementační model se zaměřuje na „grafické zobrazení implementovaného kódu“. [Buch2007]

**Obrázek 8:** Porovnání třídy analytického a návrhového modelu tříd




Zdroj: http://uml.czweb.org/diagram_trid.htm

## 4.2 PRVKY DIAGRAMU TŘÍD

Mezi prvky používané v diagramu tříd lze zařadit třídy (classes), asociace (associations), rozhraní (interfaces) a popřípadě balíčky (packages). [Buch2007] Třída je „abstrakcí objektů se stejnými vlastnostmi, stejným chováním a stejnými vztahy k ostatním objektům.“ [Buch2007] Každá třída má popsány vlastnosti a operace (souhrnně označovány jako features), které může provádět, a omezení definující, jak mohou být jednotlivé třídy propojeny. [Fow2003] Vlastnosti tříd jsou v UML označovány jako atributy, operace ve fázi návrhu jako metody (V rámci analýzy se používá označení operace). Třídy jsou vzájemně propojeny pomocí asociací. [Buch2007]; [Arl2008]

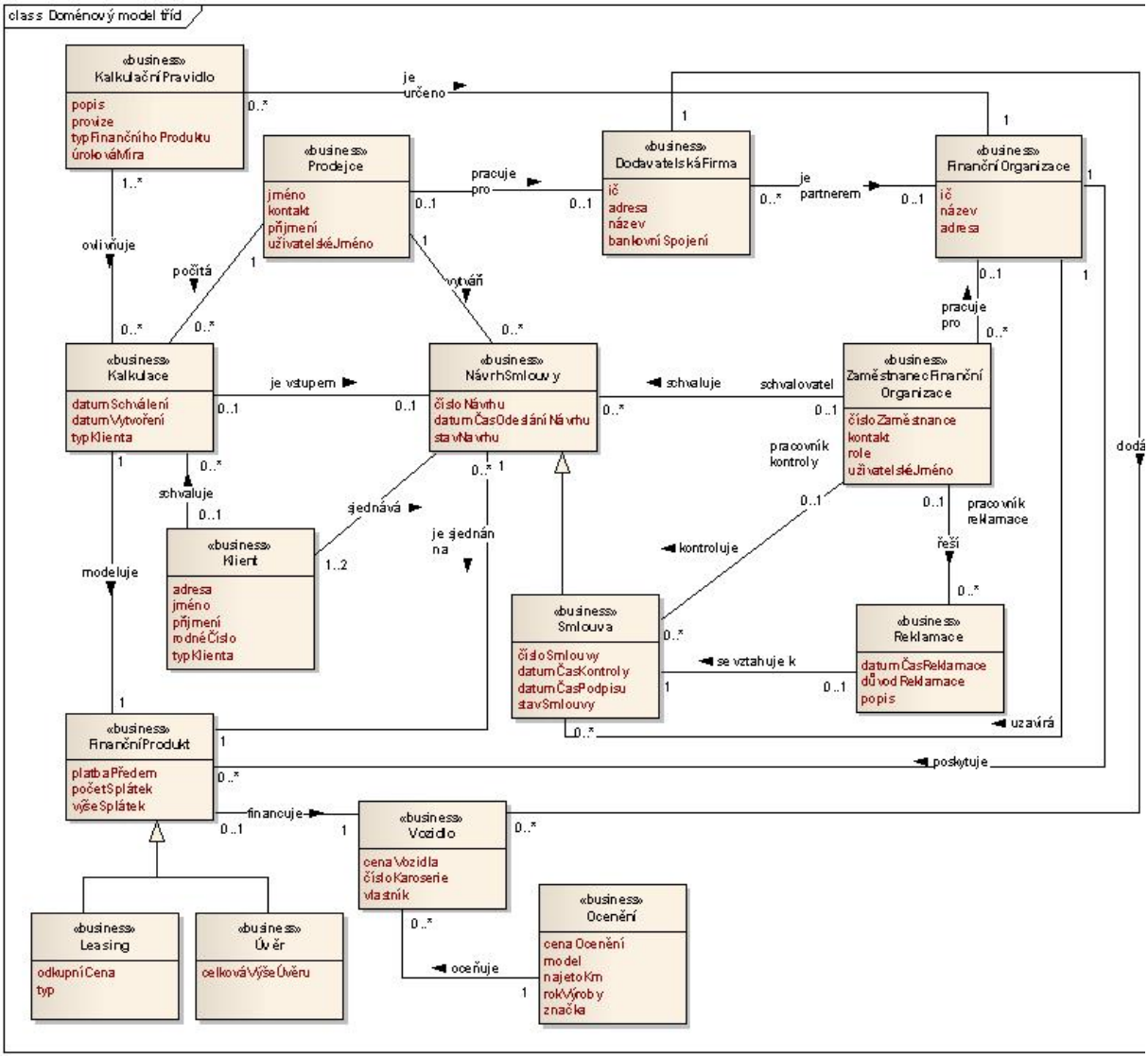
## 4.3 DOPORUČENÍ K VYTVÁŘENÍ MODELU TŘÍD

Dle [Clas2006b] a [Arl2008] platí pro diagramy tříd následující doporučení:

1. Odpovědnosti tříd, zaznamenávané jako atributy, by měly být identifikovány již v doménovém (analytickém) modelu. Každá analytická třída by měla mít tři až pět odpovědností.
2. Pro názvy tříd měla být použita běžná terminologie problémové domény. Obvykle se preferují podstatná jména v jednotném čísle.
3. Názvy tříd by měly být uváděny v tzv. VelbloudíNotaci. Pomocí této notace by měly být nazývány i metody a atributy s tím rozdílem, že jejich názvy by měly začínat malým písmenem.
4. Třídy by neměly být pojmenovávány zkratkami.
5. Asociační třídy je možné modelovat pouze v doménovém modelu tříd. Asociace, které mají asociační třídy, by neměly být pojmenovány.
6. Konceptuální model je vhodné vytvářet tak, aby byl implementačně nezávislý.
7. Designový model by měl korespondovat se jmennými konvencemi použitého programovacího jazyka (např. Java programming conventions).
8. Atributy a metody by měly být vždy setříděny dle viditelnosti od těch s největší viditelností po ty s viditelností nejmenší.
9. Asociace by měly mít vždy vyznačeny násobnosti.
10. U asociací by měl být vyznačen směr čtení.
11. Při modelování by měl být kladen důraz na minimalizaci vazeb mezi třídami. Třída by měla být spojena pouze s tolika dalšími třídami, aby jí bylo umožněno realizovat to, za co je odpovědná.

## 4.4 PŘÍKLADOVÝ MODEL TŘÍD

**Obrázek 9:** Doménový model tříd




Zdroj: http://uml.czweb.org/diagram_trid.htm

## SHRNUTÍ

Tato kapitola vysvětluje pojmy objekty a třídy. Jsou v ní obsaženy doporučení pro jejich vytváření.

## LITERATURA

Text byl převzat z literatury:

http://uml.czweb.org/diagram_trid.htm

**[Amb2004]** AMBLER, Scott W. The Object Primer: Agile Model-Driven Development with UML 2.0. 3. edition. Cambridge University Press, 2004. 572 s. ISBN 0521540186.

**[Arl2003]** ARLOW, J. NEUSTADT, I. UML a unifikovaný proces vývoje aplikací: Objektově orientovaná analýza a návrh prakticky. Přel. Bogdan Kizska. Brno: Computer Press, 2008. 387 s. ISBN 80-7226-947-X.

**[Fow2003]** FOWLER, M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3. edition. Addison Wesley, 2003. 208 s. ISBN 0- 321-19368-7.

**[Schm2001]** SCHMULLER, J. Myslíme v jazyku UML : knihovna programátora. přel. Jiří Hynek. 1.vyd. Praha : Grada, 2001. 360 s. ISBN 80-247-0029-8.

**[Arl2008]** ARLOW, J. NEUSTADT, I. UML 2 a unifikovaný proces vývoje aplikací: Objektově orientovaná analýza a návrh prakticky. Pěel. Bogdan Kizska. 1.vyd. Brno: Computer Press, 2008. 567 s. ISBN 978-80-251- 1503-9.

**[Pen2004]** PENDER, T. UML Bible. 1.vyd. Indianapolis: Wiley Publishing, Inc., 2003. ISBN 0-7645-2604-9.

# 5 POPIS JAZYKA UML – MODEL OBJEKTOVÉ SPOLUPRÁCE

## 5.1 ZÁKLADNÍ CHARAKTERISTIKA OBJEKTOVÉHO DIAGRAMU

Objektový diagram (Object Diagram) je snímkem objektů a jejich vztahů v systému v určitém časovém okamžiku. [Buch2007] Z důvodu, že zobrazuje instance tříd, je také nazýván diagramem instancí. Používá se především pro znázornění určité konfigurace objektů či zobrazení vzájemně propojených objektů ve speciálních situacích, kdy je diagram tříd či sekvenční diagram nepostačující. [Buch2007]; [Fow2003] Objektový diagram může být chápán jako speciální případ diagramu tříd vytvářený za účelem zdůraznit vazby mezi instancemi.[5]
Objektový diagram je užitečný také v počátečních fázích projektu pro modelování ukázek problémové domény, které odhalují objekty a jejich vztahy (viz. obrázek 10). Často se také používá pro modelování testovacích případů (test cases) pro ověření správnosti diagramu tříd. [Pen2003]--

## 5.2 PRVKY OBJEKTOVÉHO DIAGRAMU

Objektový diagram se svou notací velmi podobá diagramu tříd či komunikace a obvykle obsahuje pouze objekty (objects) a spojení (connections) mezi nimi. Atributy se u objektů vyznačují pouze v případě, že je to nutné pro jejich jednoznačnou identifikaci, metody se neuvádějí vůbec. [5]; [Obj2006]

## 5.3 DOPORUČENÍ K VYTVÁŘENÍ OBJEKTOVÉHO DIAGRAMU

Dle [Arl2008] a [Pen2003] platí pro názvy objektů následující doporučení:

1) Pro názvy objektů by měla být používána tzv. VelbloudíNotace, přičemž názvy by měly začínat malým písmenem.

2) Názvy objektů by měly být vždy podtrženy a nejčastěji ve tvaru Objekt:Třída (resp. :Třída).

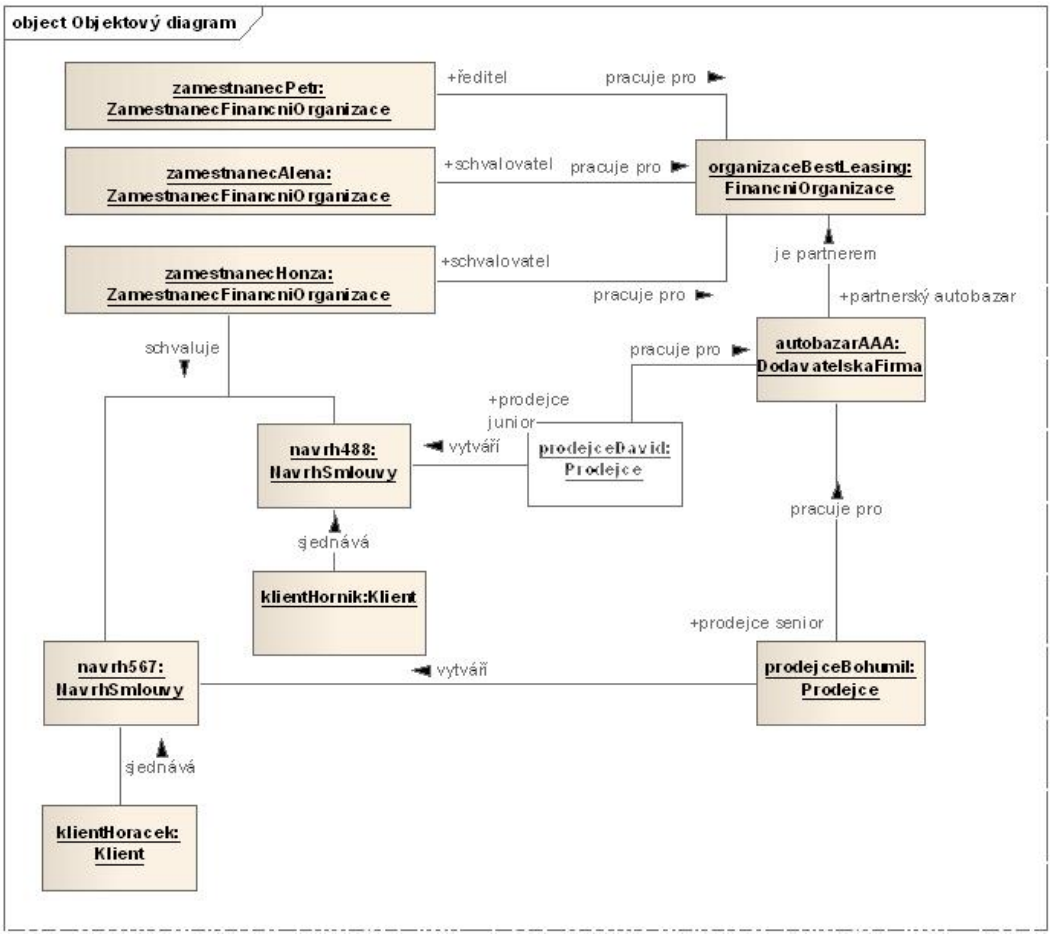
3) Role objektů by měly být uvedeny pouze v případě, že nevyplývají z názvu objektu či spojení.

4) Uvádění násobností u spojení není nutné.

5) N-ární spojení, která mohou propojit více než dva objekty, by neměla být příliš často používána.

## 5.4 PŘÍKLADOVÝ OBJEKTOVÝ DIAGRAM

**Obrázek 10:** Objektový diagram




Zdroj: http://uml.czweb.org/diagram_trid.htm

## SHRNUTÍ

V této kapitole byla probrána metodika vytváření objektového diagramu.

## LITERATURA

http://uml.czweb.org/diagram_trid.htm

**[Buch2007]** BUCHALCEVOVÁ, Alena; PAVLÍČKOVÁ, Jarmila; PAVLÍČEK, Luboš. Základy softwarového inženýrství – materiály ke cvi4ení. 1.vyd. Praha: Vysoká škola ekonomická, 2007. 222 s. ISBN 987-80-245-1270-9.

**[Fow2003]** FOWLER, Martin. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3. edition. Addison Wesley, 2003. 208 s. ISBN 0- 321-19368-7.

**[Kan2004]** KANISOVÁ, Hana; Miller, Miroslav. UML srozumitelně, 1.vyd, Brno, Computer Press, 2004. 158 s. ISBN 80-251-0231-9.

# 6 POPIS JAZYKA UML – STAVOVÉ DIAGRAMY

## 6.1 ZÁKLADNÍ CHARAKTERISTIKA

Stavový diagram (State Machine Diagram) *„zachycuje jednotlivé stavy objektu a přechody mezi nimi.“* [Buch2007]. Stavové diagramy se používají především pro popis chování určitého objektu napříč více případy užití a jejich vznik je spojen už s prvními objektově orientovanými technikami. [Fow2003]

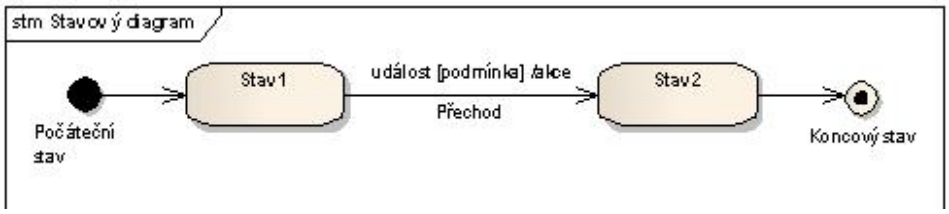
## 6.2 PRVKY STAVOVÉHO DIAGRAMU

Základními prvky stavového diagramu jsou stavy, přechody a události. Pokud to CASE nástroj umožňuje, může být diagram ohraničen rámem s názvem objektu. V případě, že se stavy nepohybují v cyklu, měl by diagram obsahovat **počáteční** (initial state) a **koncový stav** (final state). [Arl2008]

**Stav** (state) je dle [Rum2004] „situace v životě objektu, během níž objekt splňuje nějakou podmínku, provádí nějakou operaci nebo čeká na událost.“

**Přechody** (transitions) představují podmínky pro přechod objektu z jednoho stavu do druhého. V diagramu jsou značeny linií vedoucí od jednoho stavu k druhému. Jejich popis se skládá ze tří základních částí: událost [podmínka]/ akce. K vykonání uvedené akce a přechodu do dalšího stavu může dojít pouze v případě, pokud je při vzniku události uvedená **podmínka** (guard) pravdivá. **Událost** (trigger signature) je *„specifikací určitého výskytu něčeho v čase a prostoru.“* [Arl2008] Pokud není v diagramu uvedena, znamená to, že přechod do dalšího stavu probíhá automaticky. Na obrázku 19 jsou přechody označeny pouze událostmi (např. posuzování zahájeno). [Arl2008]; [Fow2003].

**Obrázek 11** Prvky stavového diagramu




Zdroj: http://uml.czweb.org/diagram_trid.htm

## 6.3 DOPORUČENÍ K VYTVÁŘENÍ STAVOVÉHO DIAGRAMU

Dle [Sta2006] a [Arl2008] platí pro stavové diagramy následující doporučení:

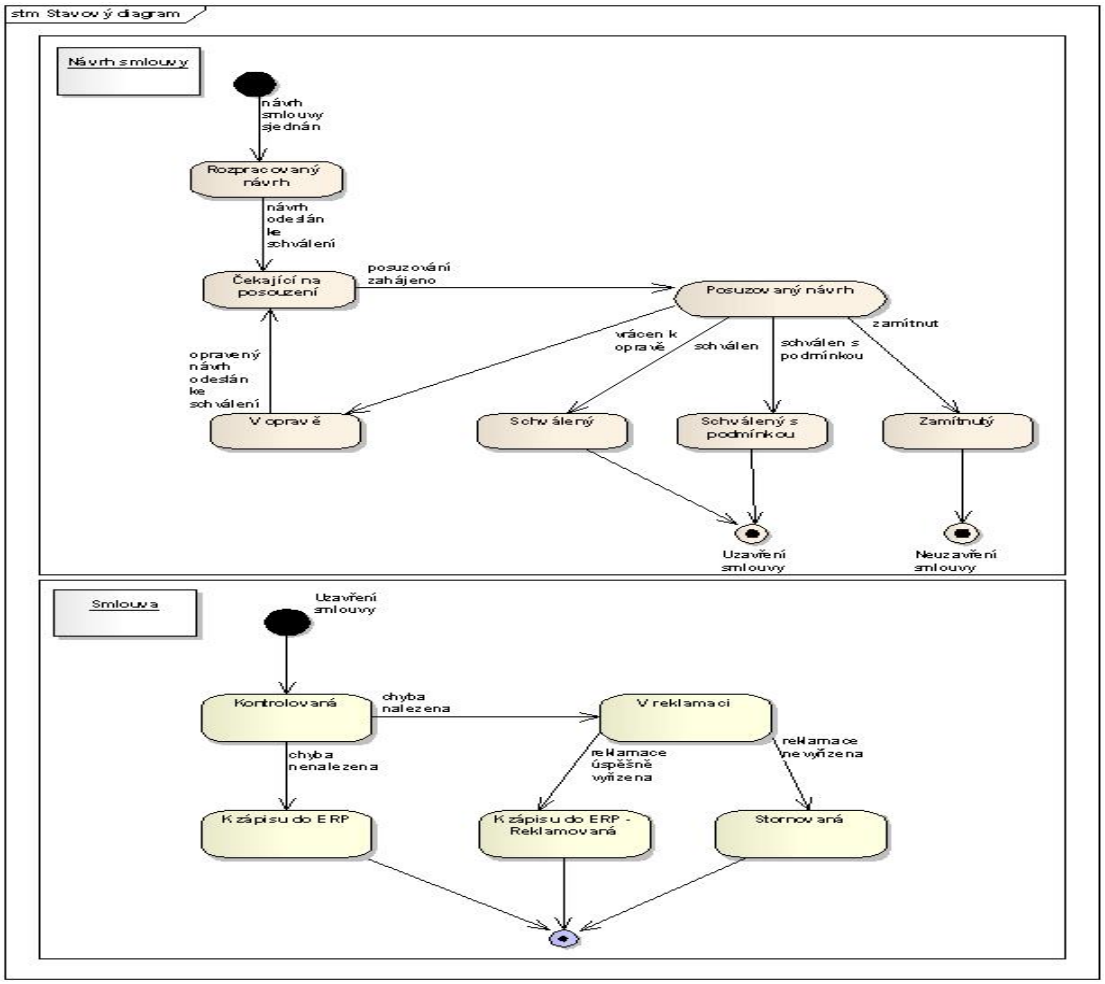
- Počáteční stavy by měly být umístěny v levém horním rohu diagramu, konečné stavy v pravém dolním.
- S výjimkou počátečních a koncových stavů by měly mít všechny stavy jak výstupní tak vstupní přechod.
- Stavy by měly být pojmenovány jednoduchým popisným názvem.

- Události je vhodné pojmenovávat slovesem v trpném rodě.
- Pro lepší čitelnost diagramu je vhodné označení přechodu umísťovat u zdrojového stavu:
    - nad přechod vedoucí zleva doprava
    - pod přechod vedoucí zprava doleva
    - zprava přechodu vedoucího seshora dolů
    - zleva přechodu vedoucího zdola nahoru

Pro podmínky přechodu platí stejná doporučení jako pro podmínky rozhodovacích uzlů diagramu aktivit.

## 6.4 PŘÍKLADOVÝ STAVOVÝ DIAGRAM

**Obrázek 12:** Stavový diagram




Zdroj: http://uml.czweb.org/

## SHRNUTÍ

V této kapitole byl popsán stavový diagram.

## LITERATURA

V této kapitole byla probrána metodika vytváření stavového diagramu. Bylo při tom čerpáno z literatury:

**[Buch2007]** BUCHALCEVOVÁ, A. PAVLÍČKOVÁ, J. PAVLÍČEK, L. Základy softwarového inženýrství - materiály ke cvi4ení. 1.vyd. Praha: Vysoká škola ekonomická, 2007. 222 s. ISBN 987-80-245-1270-9.

**[Fow2003]** FOWLER, M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3. edition. Addison Wesley, 2003. 208 s. ISBN 0- 321-19368-7.

**[Kan2004]** KANISOVÁ, H. Miller, M. UML srozumitelně, 1.vyd, Brno, Computer Press, 2004. 158 s. ISBN 80-251-0231-9.

# 7 POPIS JAZYKA UML – DIAGRAMY AKTIVIT

## 7.1 ZÁKLADNÍ CHARAKTERISTIKA DIAGRAMU AKTIVIT

Diagram aktivit (Activity Diagram) je typem diagramu interakcí, který se používá pro popis procedurální logiky, byznys procesů (viz obrázek 17) či pracovních postupů. Umožňuje také graficky modelovat jednotlivé případy užití jako posloupnost akcí (viz obrázek 15). [Fow2003] [Arl2008]

Diagram aktivit prodělal od svého vzniku mnoho úprav a i s novou verzí UML 2.0 došlo k jeho změnám. Zatímco v UML 1.x byl chápan jako speciální případ stavového diagramu, UML 2.0 už tyto dva diagramy nijak nespojuje. [Fow2003] Diagram získal novou sémantiku založenou na formálním grafickém modelovacím jazyce Petri Nets (Petriho sítě). [Arl2008]

## 7.2 PRVKY DIAGRAMU AKTIVIT

Diagram aktivit modeluje procesy jako aktivity, které se skládají z **uzlů** (nodes) vzájemně propojených **hranami** (edges). [ARL2007]

Existují tři typy uzlů – **akční uzly**, které reprezentují samostatné a v rámci aktivity nedělitelné jednotky, **řídící uzly**, jejichž úkolem je řídit cestu uvnitř aktivity a **uzly objektové**, které zastupují objekty. Nejpoužívanějším akčním uzlem je tzv. call action node, který inicializuje aktivitu, chování či operaci. Příkladem řídících uzlů jsou počáteční (initial nodes), konečné uzly (final nodes) nebo uzly rozhodnutí (decision nodes). [Arl2008]

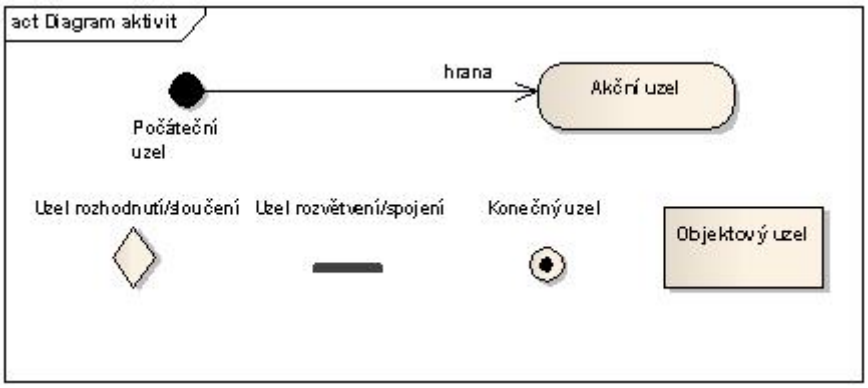
**Uzel rozhodnutí** (v UML 1.x dříve označovaný *branch*) má jednu vstupní hran u aněkolik konkurujících si hran výstupních. Daný výstup bude zvolen podle toho, která ze vzájemně se vylučujících kontrolních podmínek bude splněna. K označení hrany, která bude použita, pokud nebude splněna žádná kontrolní podmínka, se používá klíčové slovo *jinak* (else). [Fow2003]; [Arl2008]

**Uzel sloučení** (merge) může mít několik vstupních, ale pouze jednu výstupní hranu. Používá se především pro sjednocení větví diagramu aktivit, které byly předtím rozděleny uzlem rozhodnutí. [Arl2008]

Procesy, které diagram popisuje, mohou probíhat paralelně, což je umožněno řídícími uzly **rozvětvení** (fork) a **spojení** (join). [Fow2003]

Pro zpřehlednění se může diagram rozdělit například dle rolí (viz obrázek 15) či organizačních jednotek do tzv. **zón odpovědnosti** či **plaveckých drah** (swimlanes). [Arl2008] Pro znázornění zón odpovědnosti nabízí UML 2.0 i textovou notaci (viz obrázek 17), která se ale používá pouze v případě, kdy by rozdělení diagramu do zón zhoršilo jeho přehlednost. Obvykle bývá grafický zápis daleko srozumitelnější.

**Obrázek 13:** Prvky diagramu aktivit



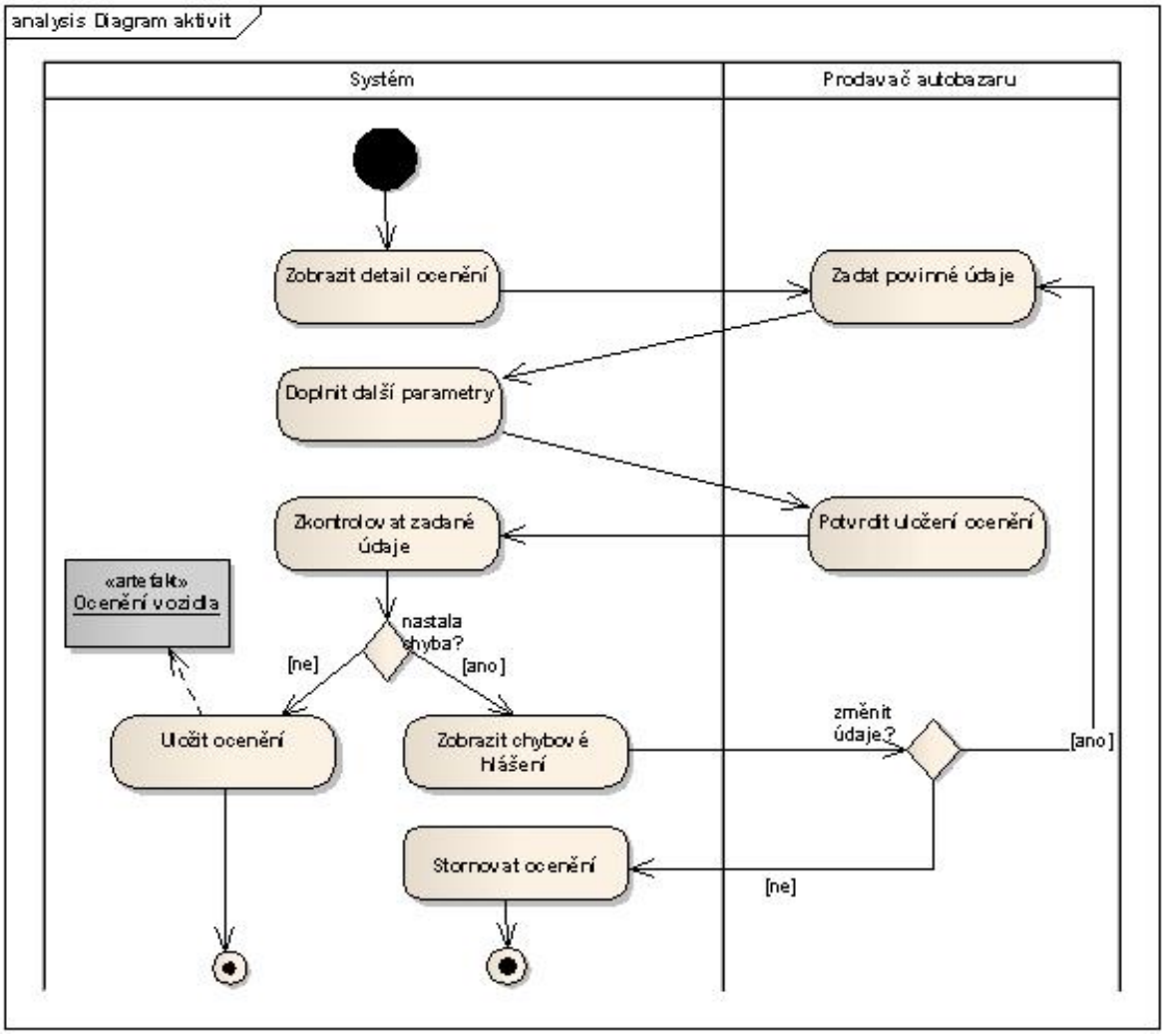

## 7.3 DOPORUČENÍ K VYTVÁŘENÍ DIAGRAMU AKTIVIT

Dle [Act2006] platí pro diagramy aktivit následující doporučení:

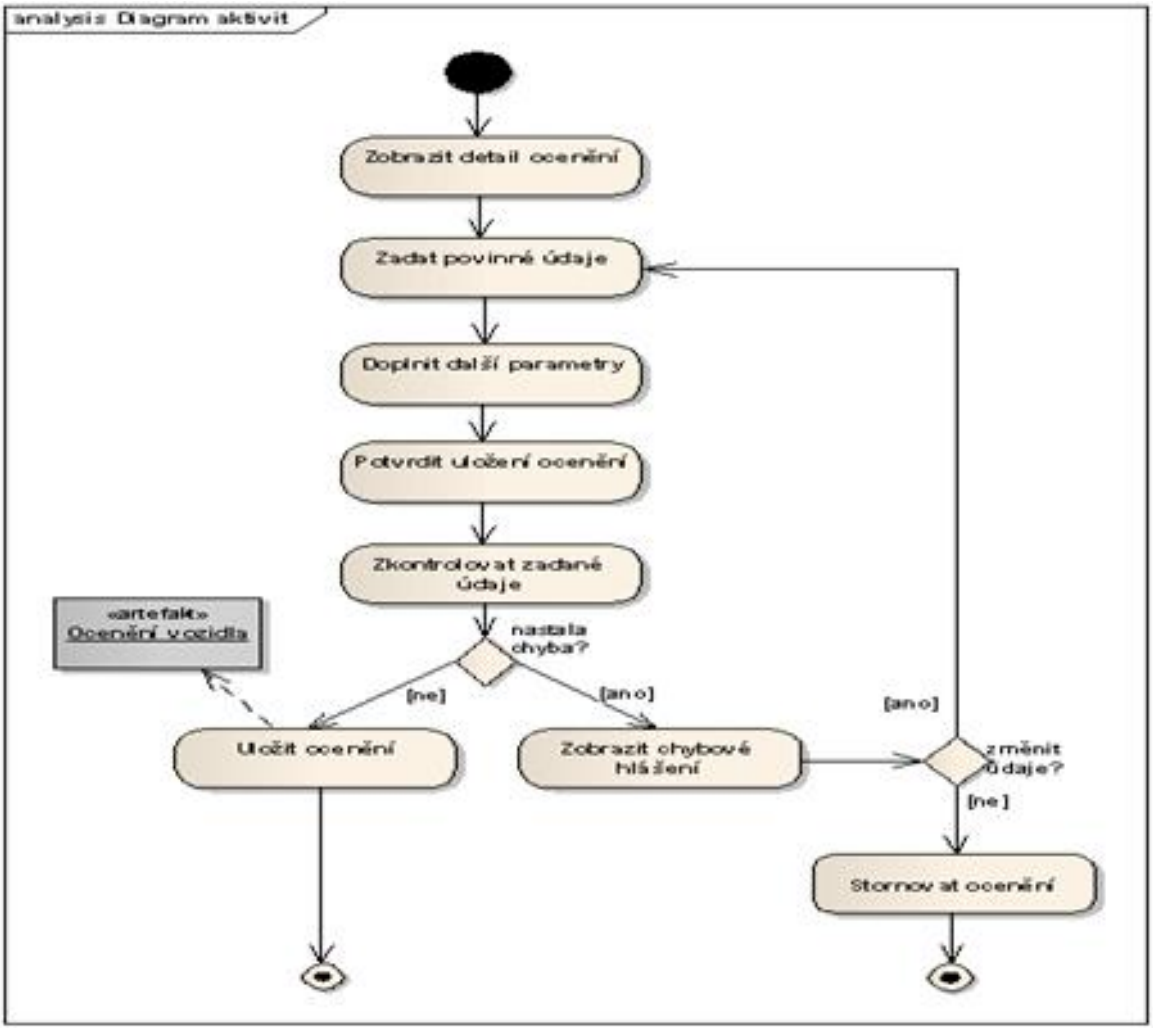
- Kontrolní podmínky rozhodovacích uzlů by se měly vždy vylučovat a měly by zahrnovat všechny možnosti, které mohou nastat. Nesprávné podmínky jsou například x <= 9 a x >= 9, které se překrývají, a není tedy možné jednoznačně určit, co se stane v případě, že x = 9.
- Počáteční uzly by měly být umístěny v levém horním rohu diagramu.
- Diagram aktivit by měl vždy obsahovat konečný uzel, pokud se nejedná o opakující se proces.
- Každý akční uzel by měl mít jak vstupní, tak výstupní hranu.
- Diagram by neměl obsahovat více jak pět plaveckých drah.
- Plavecké dráhy by měly být znázorněny vertikálně. Jejich horizontální orientace je nevhodná především při modelování byznys procesů.
- Pokud se v diagramu vyskytuje více identických objektových uzlů (objektů), je vhodné je odlišit vyznačením stavů (např. Objednávka [otevřená], Objednávka [zavřená]).

## 7.4 PŘÍKLADOVÝ DIAGRAM AKTIVIT

**Obrázek 14:** Použití plaveckých drah v diagramu aktivit (grafický zápis)

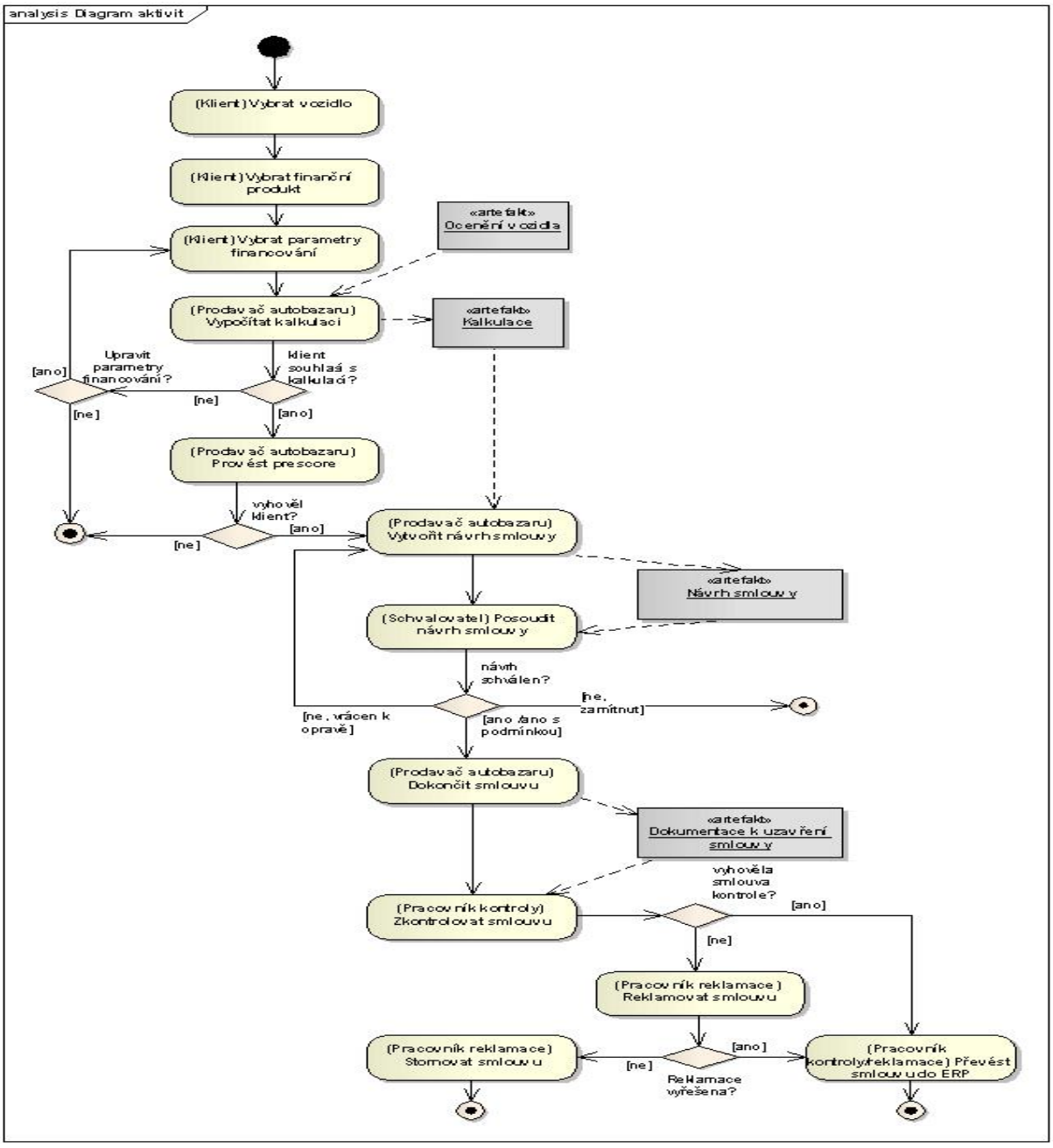


**Obrázek 15:** Diagram aktivit ocenění




Zdroj: http://uml.czweb.org/

**Obrázek 16:** Diagram aktivit autobazar – vytvoření smlouvy




Zdroj: http://uml.czweb.org/

## SHRNUTÍ

V této kapitole byla probrána metodika vytváření diagramu aktivit.

## LITERATURA

http://uml.czweb.org/

**[Buch2007]** BUCHALCEVOVÁ, A. PAVLÍČKOVÁ, J. PAVLÍČEK, L. Základy softwarového inženýrství - materiály ke cvi4ení. 1.vyd. Praha: Vysoká škola ekonomická, 2007. 222 s. ISBN 987-80-245-1270-9.

**[Fow2003]** FOWLER, M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3. edition. Addison Wesley, 2003. 208 s. ISBN 0- 321-19368-7.

**[Kan2004]** KANISOVÁ, H. Miller, M. UML srozumitelně, 1.vyd, Brno, Computer Press, 2004. 158 s. ISBN 80-251-0231-9.

# 8 SEKVENČNÍ DIAGRAM

## 8.1 ZÁKLADNÍ CHARAKTERISTIKA SEKVENČNÍHO DIAGRAMU

Sekvenční diagram (Sequence Diagram) je nejvíce používaným diagramem interakcí. *„Zachycuje grafický průběh zpracování v systému v podobě zasílání zpráv.“* [Buch2007]

Sekvenční diagram nejčastěji zobrazuje chování a spolupráci jednotlivých objektů v rámci jednoho případu užití. Pro popis chování jednoho objektu napříč více případy užití se používá stavový diagram. [Fow2003]

Prvky diagramu

**Zprávy** mohou být v sekvenčním diagramu posílány jak mezi jednotlivými objekty, tak i třídami či dokonce aktéry. Proto se prvky, které mezi sebou v diagramu komunikují, nazývají souhrnně **klasifikátory** (classifiers). [Buch2007] Z každého klasifikátoru vede tzv. **čára života** (lifeline), která reprezentuje, jakým způsoben se instance určitého klasifikátoru účastní interakce. [Arl2008]

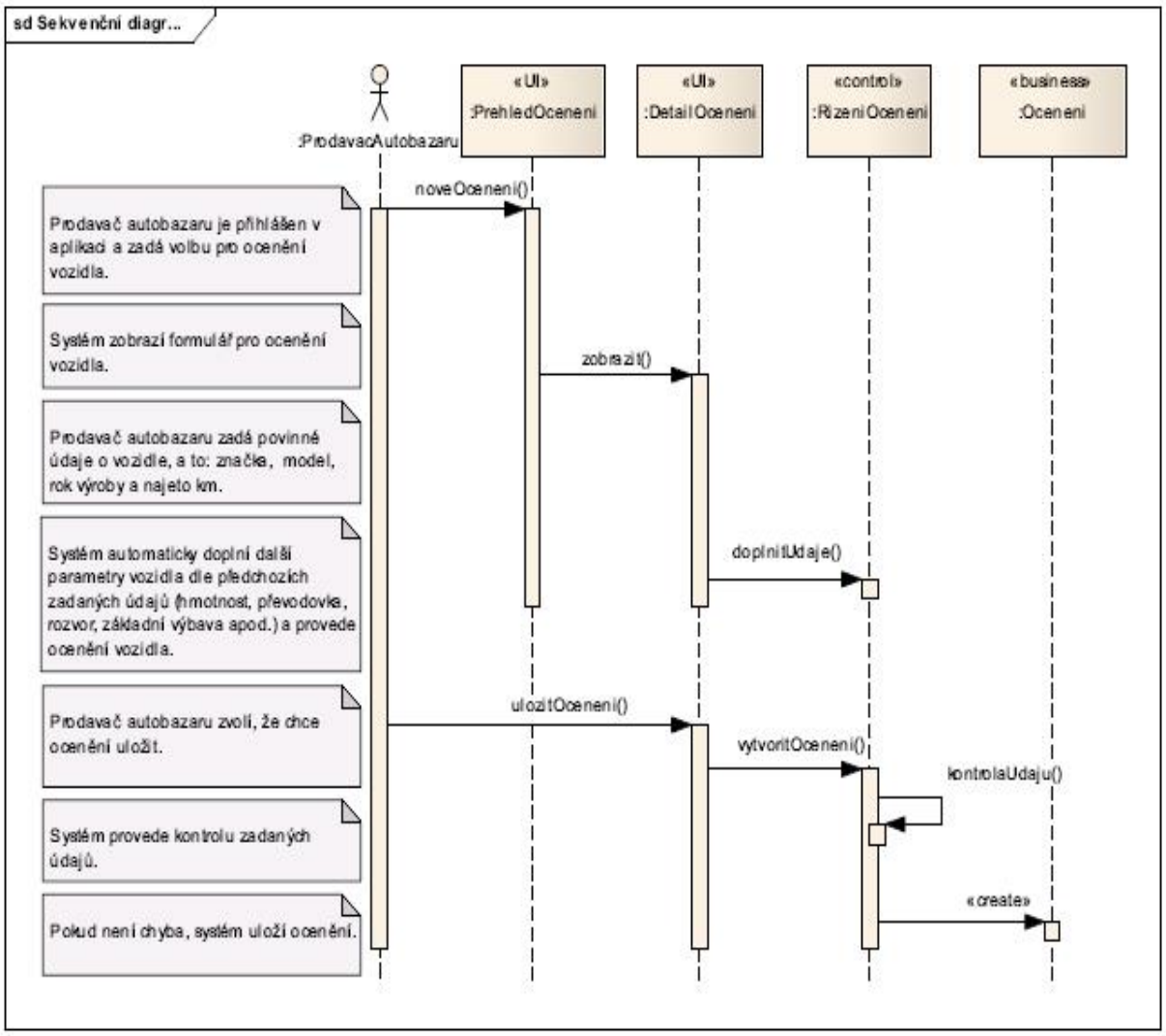
## 8.2 DOPORUČENÍ K VYTVÁŘENÍ SEKVENČNÍHO DIAGRAMU

Dle [Seq2006], [Arl2008] a [Buch2007] platí pro sekvenční diagram následující doporučení:

- Pokud je to možné, měly by být zprávy orientovány zleva doprava.
- Objekty by se měly pojmenovávat pouze v případě, že na ně chceme v diagramu odkazovat, nebo když v diagramu existuje více objektů stejného typu.
- Aktéři by měli být pojmenováni stejnými názvy jako v diagramu případů užití.
- Třídy by měly být pojmenovány stejnými názvy jako v diagramu tříd. Aktér se může jmenovat stejně jako třída.
- Zprávy by měly být vždy pojmenovány či alespoň označeny stereotypem. Názvy zpráv by měly odpovídat názvům metod, které jsou vyvolány na cílovém klasifikátoru. Pokud je tímto klasifikátorem aktér, postačí jako název krátká charakteristika komunikace.
- Návratové zprávy se obvykle v diagramu nemodelují.
- Diagram by měl být doplněn popisem, což značně zvyšuje jeho srozumitelnost.
- Pokud se ve více sekvenčních diagramech vyskytuje stejná posloupnost zpráv (interakce), je možné použít tzv. **mechanismus výskytu interakce**. Výskyty interakce odkazují na určitou, již dříve vytvořenou interakci, a není tedy nutné ji stále celou překreslovat do nových diagramů.

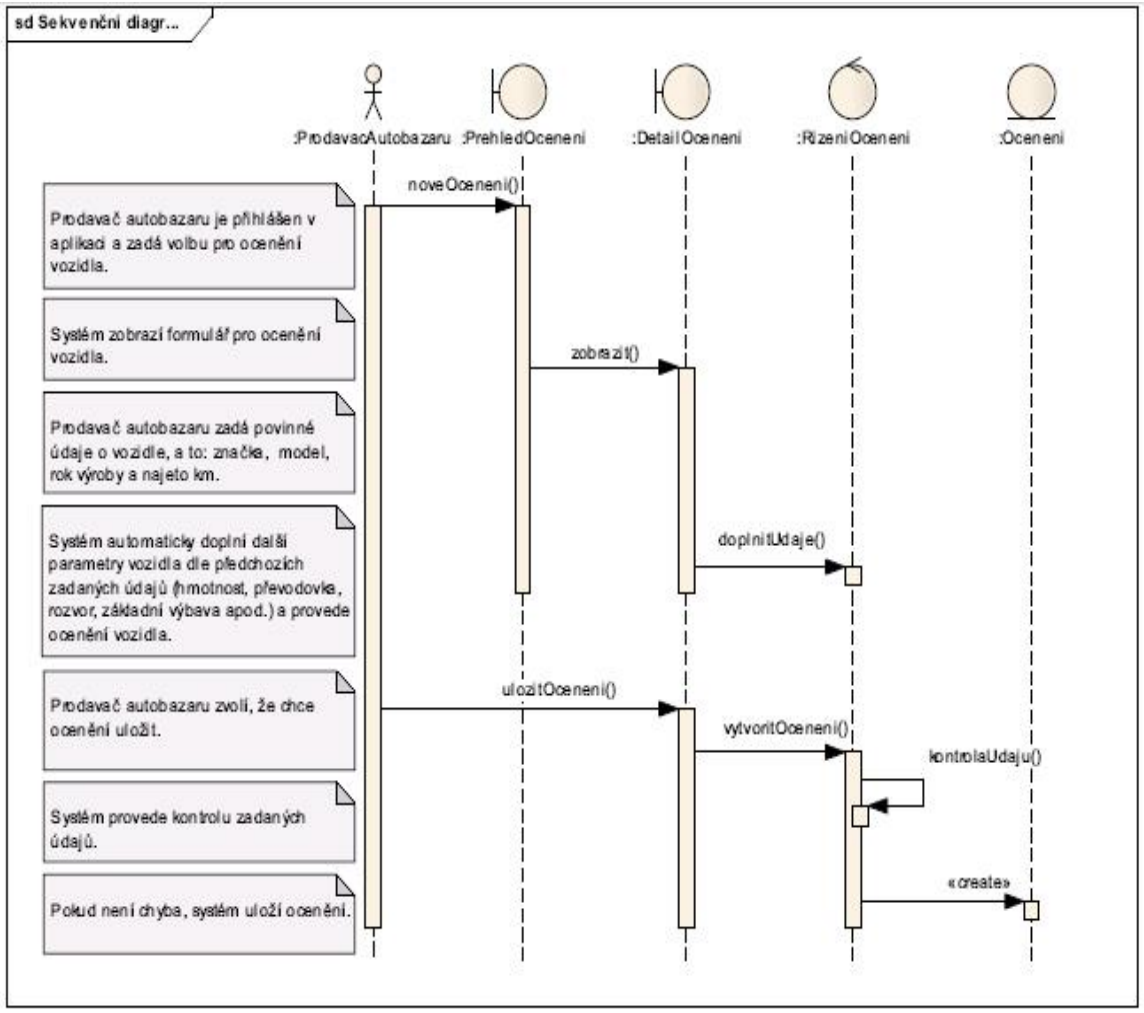
## 8.3 PŘÍKLADOVÝ SEKVENČNÍ DIAGRAM

**Obrázek 17:** Sekvenční diagram popisující vytvoření ocenění vozidla




Zdroj: http://uml.czweb.org/

**Obrázek 18:** Sekvenční diagram ocenění (grafické znázornění stereotypů)




Zdroj: http://uml.czweb.org/

## SHRNUTÍ

V této kapitole byla probrána metodika vytváření sekvenčního diagramu.

## LITERATURA

**[Buch2007]** BUCHALCEVOVÁ, A. PAVLÍČKOVÁ, J. PAVLÍČEK, L. Základy softwarového inženýrství - materiály ke cvi4ení. 1.vyd. Praha: Vysoká škola ekonomická, 2007. 222 s. ISBN 987-80-245-1270-9.

**[Fow2003]** FOWLER, M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3. edition. Addison Wesley, 2003. 208 s. ISBN 0- 321-19368-7.

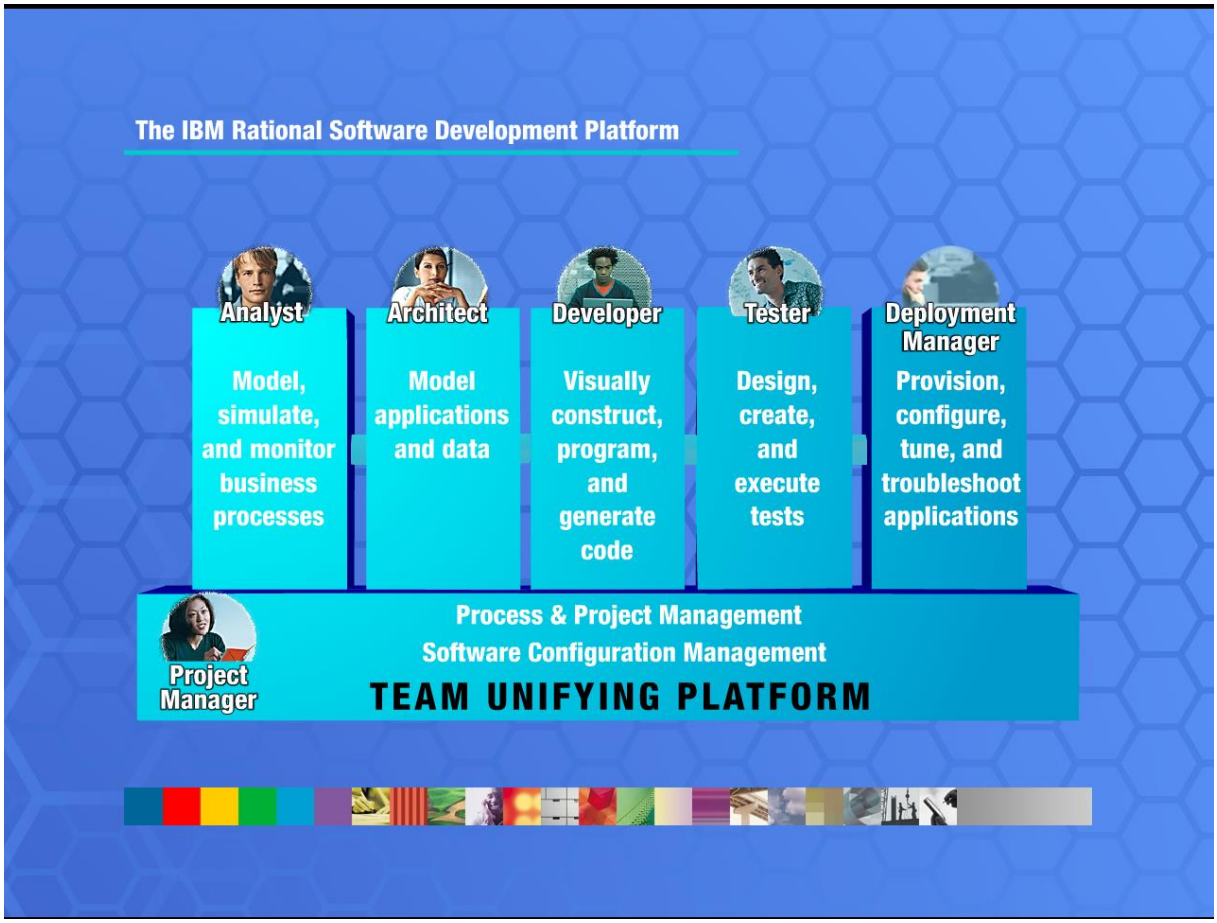
**[Kan2004]** KANISOVÁ, H. Miller, M. UML srozumitelně, 1.vyd, Brno, Computer Press, 2004. 158 s. ISBN 80-251-0231-9.

# 9 PŘEHLED SOFTWARE PRODUKTŮ PRO PRÁCI S UML

## 9.1 IBM RATIONAL SOFTWARE DEVELOPMENT PLATFORM

Case nástroj firmy IBM pro poporu vývoje aplikací zahrnuje všechny možné nástroje od popisu firemních procesů až po kreslení diagramů UML. Je určen pro týmovou práci při řešení rozsáhlých projektů vyvíjejících informační systém.

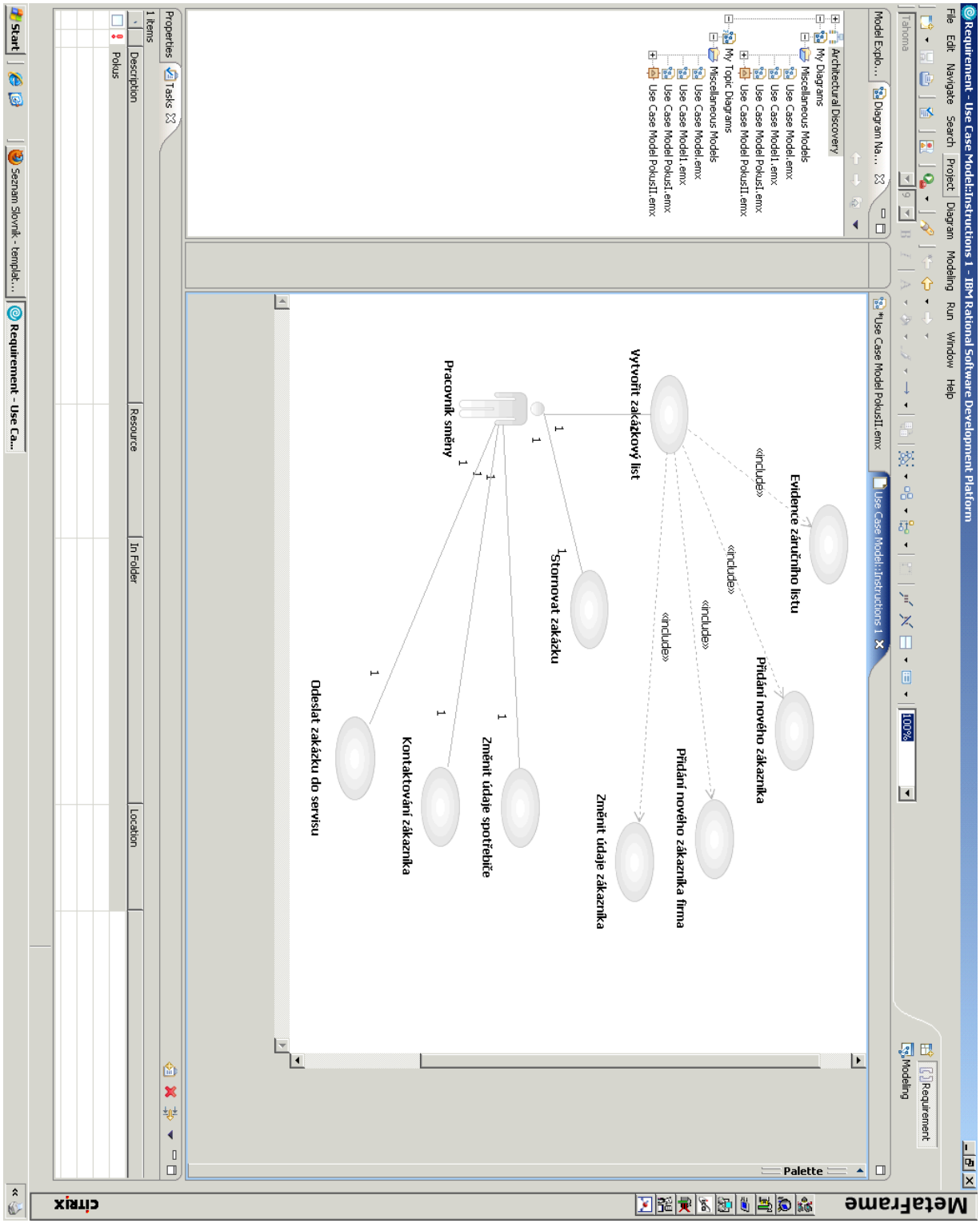
**Obrázek 19: Týmová práce při vývoji v software IBM Rational development platform**




Zdroj: Obrazovka ze software IBM Rational Development Platform

Tento CASE nástroj je nainstalován na serveru a je dostupný z počítačových učeben Obchodně podnikatelské fakulty v Karviné. Program je možno využit i přes vzdálený přístup tenkým klientem v systému CITRIX.

**Obrázek 20:** Příklad vývojového prostředí IBM Developer ptatform – USE CASE Model

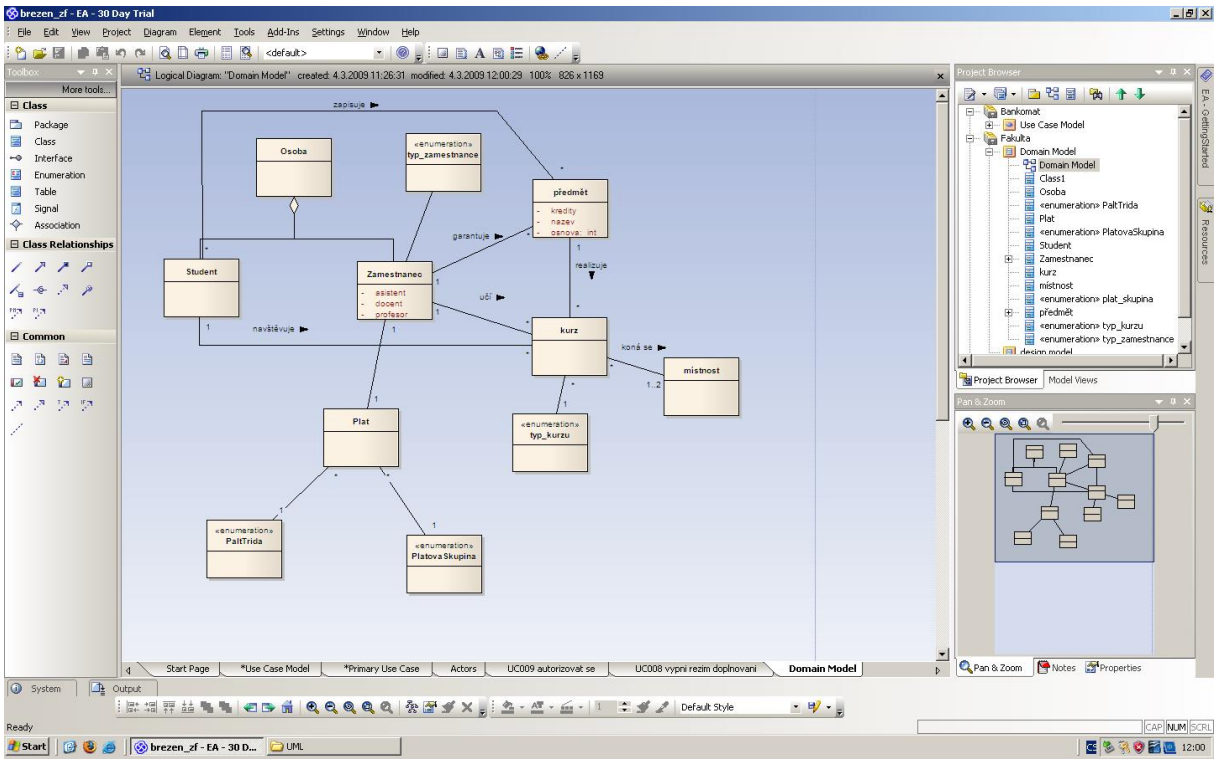



Zdroj: Vlastní s využitím software IBM Rational development platform

## 9.2 ENTERPRISE ARCHITECT FIRMY SPARX

Velmi používaný program pro kreslení UML diagramů je Enterprise Architect firmy Sparx. Program je běžně využíván pro výuku pro transparentnost, přehlednost a poměrně jednoduché ovládání.

**Obrázek 21:** Vývojové prostředí Enterprise Architect firmy Sparx – CLASS DIAGRAM




Zdroj: vlastní s využitím software Enterprise Architect

Firma Sparx umožňuje klientům a tedy i studentů stáhnout 100 denní Trial verzi.

## 9.3 UML A VISIO FIRMY MICROSOFT

Šablona diagramu modelu UML aplikace Microsoft Office Visio nabízí plnou podporu vytváření objektově orientovaných modelů složitých softwarových systémů.

- Diagram případu použití
- Diagram statické struktury
- Diagram balíčku
- Diagram činnosti
- Stavový diagram
- Sekvenční diagram
- Diagram spolupráce
- Diagram komponent
- Diagram zavedení

Program VISIO je dostupný studentům Obchodně podnikatelské fakulty na počítačových učebnách, s šablonou diagramů UML na PC učebně, kde probíhá výuka předmětu „Objektové metody modelování.

## SHRNUTÍ

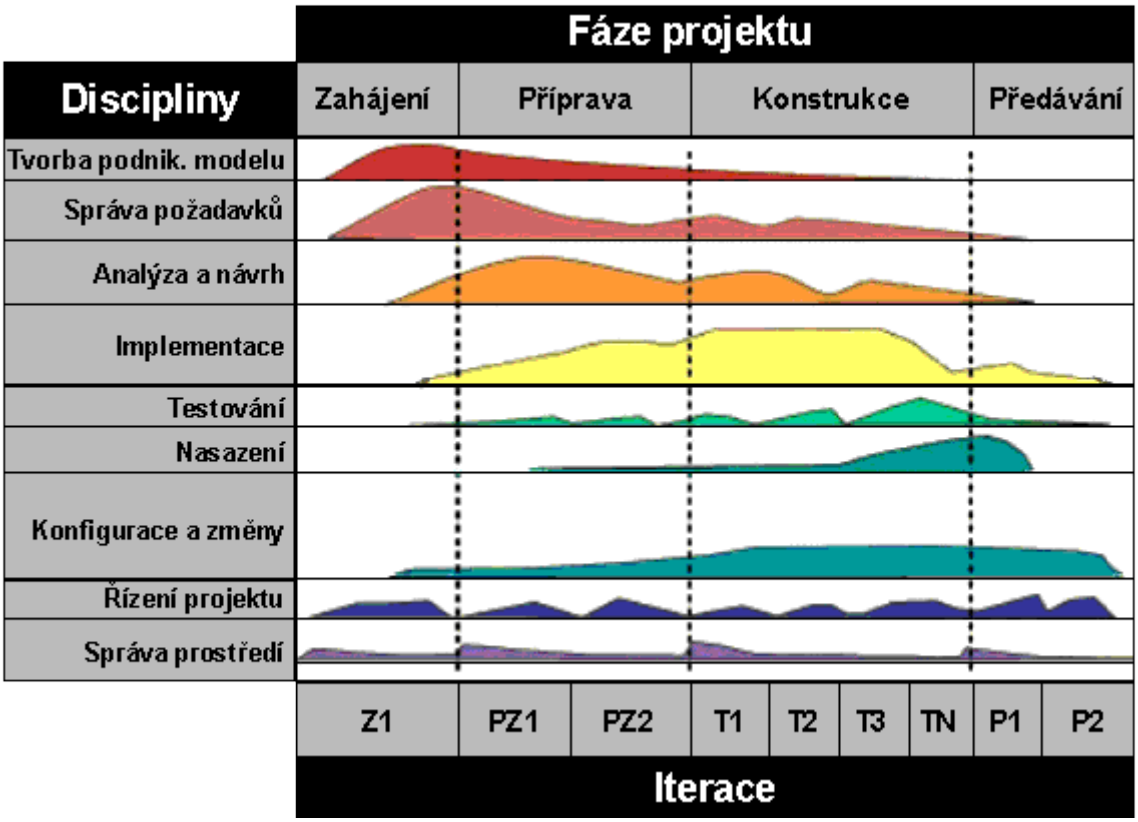
V této kapitole je uveden přehled software pro vytváření diagramů UML a využití metodiky RUP.

# 10 RUP - RATIONAL UNIFIED PROCESS

Termíne Rational Unified Process (dále též RUP) označuje metodiku vývoje software vytvořenou a používanou společností Rational Software Corporation. Společnost Rational tuto metodiku distribuovala formou softwarového produktu, je k dispozici na jejích webových stránkách (www.rational.com). Společnost Rational převzala firma IBM, ve které tvoří samostatnou divizi.

Specifika RUP názorně ilustruje následující schéma:

**Obrázek 22:** Schéma projektu dle metodiky RUP




Zdroj: ALDORF F., Metodika RUP

Horizontální osa představuje čas, na vertikální ose jsou naneseny jednotlivé discipliny, které se během projektu aplikují. Projekt je v této metodice rozdělen na čtyři fáze s danými cíli; jednotlivé fáze nejsou obvykle jednolité, nýbrž jsou prováděny v několika dílčích krocích (tzv. iteracích).

Pro modelování procesů se využívá prostředků jazyka UML (Unifed Modeling Language).

## 10.1 HISTORIE

Rational Unified Process z velké části vychází z metodiky Objectory Process, jejíž první verzi vytvořil již v roce 1987 Ivar Jacobson. Tuto metodiku postavenou na modelech případů užití a objektovém přístupu původně využívala švédská společnost Objectory AB. Tato firma se

v roce 1995 sloučila s Rational Software Corporation a došlo ke vzniku nové metodiky Rational Objectory Process 4.0.

Ta kombinuje přístup k iterativnímu vývoji a architektuře převzatý z metodiky Rational Approach s modelováním založeným na případech užití používaným v Objectory Process. Verze 4.0 z roku 1996 používala již notaci nového modelovacího jazyka UML.

Po fúzi Rational Software Corporation se společnosti SQA Inc. a Requisite Inc. vznikla verze 4.1 rozšířená o dokonalejší metody modelování požadavků.

V roce 1998, po akvizici společnosti Pure Atria, vzniká verze Rational Unified Process 5.0, která obsahuje mimo jiné zdokonalení v oblasti modelování dat, procesů a řízení projektu. Poslední verze RUP 5.5 byla uveřejněna na začátku roku 2002.

## 10.2 ZÁKLADNÍ PRAVIDLA RUP

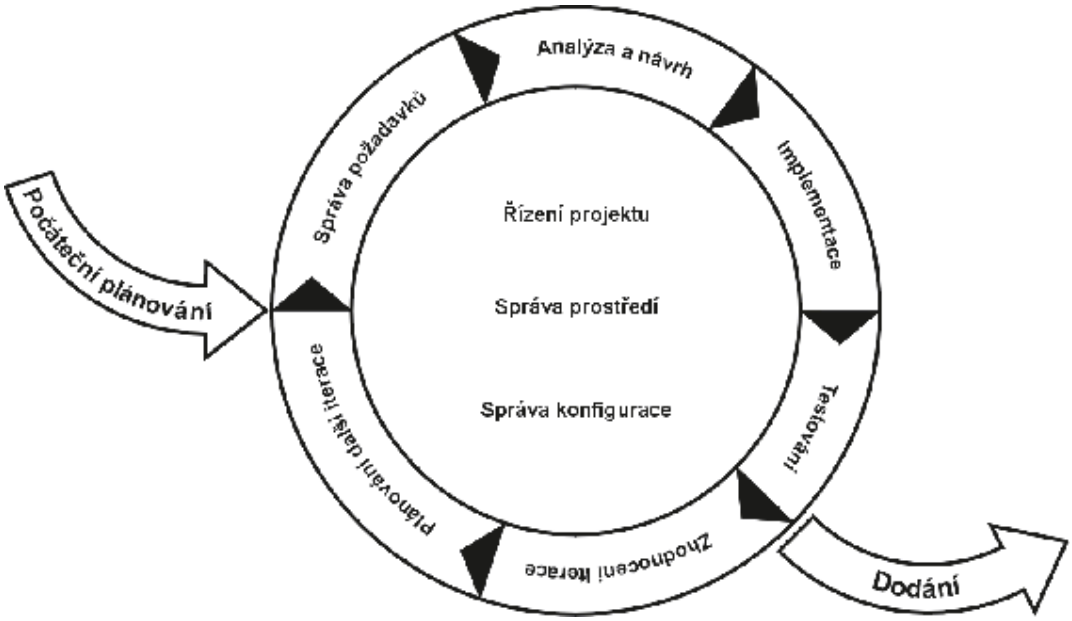
Základní filosofií metodiky Rational Unified Process je šest základních pravidel, tzv. "nejlepších praktik" používaných při vývoji software:

1. Iterativní vývoj software
2. Správa požadavků
3. Architektura založená na komponentách
4. Vizuální modelování
5. Ověřování kvality software
6. Řízení změn software

### *10.2.1 ITERATIVNÍ VÝVOJ*

U současných rozsáhlých systémů není již možné na začátku přesně definovat celý problém, navrhnout řešení, provést implementaci a až na závěr - kdy je spotřebována většina přidělených finančních prostředků - celý systém otestovat. Iterativní přístup spočívá v rozdělení celého projektu na čtyři fáze, z nichž každá se skládá z několika iterací se životním cyklem typu "vodopád". Výsledkem každé této iterace je spustitelná verze.

**Obrázek 23:** Iterativní vývoj




Zdroj: ALDORF F., Metodika RUP

Tento přístup má zejména následující výhody:

- Možnost objektivního posouzení stavu projektu
- Rovnoměrnější pracovní vytížení vývojářského týmu. Přímo souvisí s předchozím bodem. Projekt je koncipován tak, aby co nejdříve přinášel konkrétní výsledky. Při rozdělení na iterace je možné snáze sledovat průběh projektu a dodržování stanovených termínů pro jednotlivé iterace.
- Možnost testování meziverzí
- Spolupráce s uživateli v průběhu celého projektu
- Včasné rozpoznání nesrovnalostí mezi požadavky, návrhem a implementací. Tato výhoda přímo vyplývá z předchozích dvou bodů. Díky tomu, že uživatelé mají možnost kontrolovat a hodnotit dílčí části systému, značně se omezuje riziko vysokých nákladů způsobených úpravami produktu v pozdní fázi vývoje.
- Snazší zapracování změn požadavků

### *10.2.2 AKTIVNÍ SPRÁVA POŽADAVKŮ*

U vodopádového přístupu se požadavky sbírají a dokumentují pouze na začátku celého projektu, jejich změna v dalších stádiích vývoje se zpravidla nepřipouští. RUP naproti tomu využívá koncepci “aktivní správy požadavků” spočívající ve stálém kontaktu zadavatele s dodavatelem a možnosti zpřesňování a úprav požadavků v průběhu projektu.

### *10.2.3 KOMPONENTOVÁ ARCHITEKTURA*

Komponenty v tomto kontextu označují netriviální části systému zajišťující určitou jeho funkcionalitu. Využití architektury systému založené na komponentách je výhodné z několika důvodů:

- Projekt lze snáze rozdělit mezi několik vývojářských týmů

- Zapracování změn je snazší
- Umožňuje postupnou tvorbu systému

Rational Unified Process podporuje standardní komponentové infrastruktury (COM, CORBA)

### 10.2.4 VIZUÁLNÍ MODELOVÁNÍ

Pro modelování se v RUP využívá Rational Unified Process prostředků jazyka UML (detailní popis viz standard [UML]).

Grafické znázornění systému a jeho chování zpřehledňuje návrh a umožňuje udržet konzistenci mezi modelem a vlastní implementací: např. software Rational Rose je schopen na základě modelu tříd vygenerovat kód v C++. Modul "C++ analyzer" pak umožňuje převést kód zpět na grafický model tříd.

### 10.2.5 OVĚŘOVÁNÍ KVALITY SOFTWARE

Softwarový produkt lze předat do provozu pouze v případě, že splňuje požadovaná kvalitativní kritéria:

- Funkcionalita: systém podporuje všechny funkce vymezené v modelu případů užití.
- Spolehlivost: systém správně řeší určenou skupinu chybových stavů.
- Výkon: dostupnost systému a doba odezvy jsou přijatelné.

V RUP je hlavním nástrojem pro zajištění výše uvedených parametrů testování. Testování jako jedna z disciplín této metodiky.

### 10.2.6 ŘÍZENÍ ZMĚN

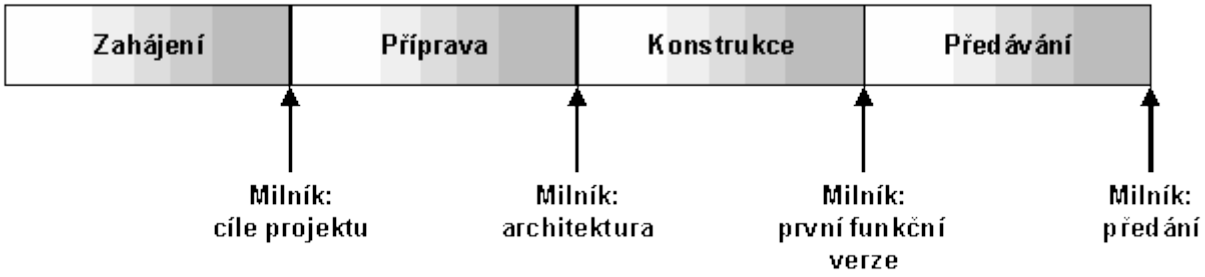
Možnost provádět změny i v pozdějších stádiích projektu s sebou samozřejmě přináší i problém jejich řízení a dokumentace.

## 10.3 ŽIVOTNÍ CYKLUS PROJEKTU: CHARAKTERISTIKA JEDNOTLIVÝCH FÁZÍ

### 10.3.1 ŽIVOTNÍ CYKLUS PRODUKTU V RUP

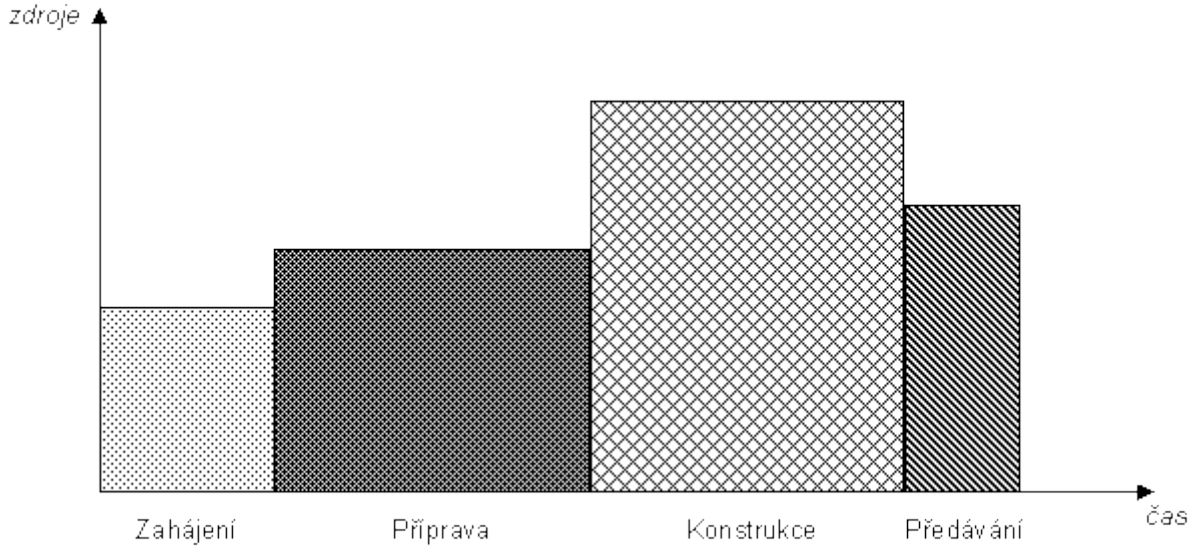
Životní cyklus projektu je v metodice Rational Unified Process rozdělen do čtyř základních fází (viz obrázek), z nichž každá je zakončena tzv. milníkem. Po dokončení každé fáze se provede zhodnocení, zda byly splněny požadované cíle. Další fázi projektu je možné zahájit pouze v případě splnění všech požadovaných kritérií.

**Obrázek 24:** Životní cyklus projektu




Zdroj: ALDORF F., Metodika RUP

**Obrázek 25:** Obvyklá náročnost jednotlivých fází (převzato z [RUP])




Zdroj: ALDORF F., Metodika RUP

Náročnost jednotlivých fází projektu na čas a zdroje závisí na typu vyvíjeného software, existenci nástrojů pro automatizaci vývoje a "generaci" software.

Vývoj produktu obvykle není definitivně ukončen uvedením první verze do provozu. Pokud software definitivně "neodumře", vytvářejí se na základě nově vznikajících potřeb jeho další verze. Vývoj následující generace produktu se z pohledu řízení projektu skládá opět ze čtyř fází. U dalších vývojových cyklů se ale zpravidla podstatně zkracuje délka prvních dvou fází, neboť analytické činnosti byly již z větší části provedeny u předchozích verzí.

## 10.4 ZAHÁJENÍ (INCEPTION)

Hlavním cílem této fáze je vymezení požadavků na systém, na němž se shodnou všechny zainteresované strany. Je třeba vymezit rozsah vytvářeného systému a pro klíčové činnosti vytvořit modely případů užití. U projektů, při nichž se pouze upravuje stávající systém, je tato fáze podstatně méně náročná, neboť výše uvedené kroky byly již dříve zčásti provedeny.

**Plán iterace** pro fázi zahájení:

1. Tvorba podnikového modelu: Základní charakteristika podniku
   Analytik podnikových procesů popíše pomocí modelů případů užití procesy, které se dotýkají vytvářeného systému.
2. Správa požadavků: Analýza problémové oblasti
   Na tomto kroku spolupracuje systémový analytik se zadavatelem. Jeho cílem je vymezit přesné určení systému a jeho rozsah. Rozhodujícím faktorem jsou požadavky uživatelů, již nyní je však třeba brát v úvahu omezení daná prostředím, do kterého budeme nový systém nasazovat (např. platformy, rozhraní na stávající systémy).
3. Správa požadavků: Identifikace požadavků zadavatele
   Na základě konzultací se zadavatelem se sepíše seznam funkcí, které by systém měl podporovat. Doporučuje se průběžně vytvářet společný slovník používaných pojmů, který usnadní tvorbu modelů případů užití a udržení jejich konzistence. V tomto kroku je již možné sestavit úvodní model případů užití systému.
4. Řízení projektu: Plánování vývoje software
   Tento krok představuje převedení vize do ekonomických dimenzí. Na základě požadavků na systém se sestaví úvodní plán vývoje software a počáteční odhad potřebných zdrojů pro další fáze (a samozřejmě s nimi souvisejících nákladů). V případě potřeby se doplní seznam rizik.
5. Správa prostředí: Příprava prostředí projektu
   Zhodnocení stavu projektu i organizace zadavatele. Procesní inženýr vytvoří první verzi Konfigurace RUP. Specialista na vývojové nástroje zvolí prostředky pro analýzu požadavků. Systémový analytik vytvoří první verzi Pravidel pro modelování případů užití.
6. Řízení projektu: Plánování vývoje software
   V tomto stádiu by již měla být jasně formulována vize systému a určena priorita mezi jednotlivými funkcemi systému. Vedoucí projektu na základě poslední verze seznamu případů užití (seřazeného podle priorit) a rizik aktualizuje plán vývoje software.
7. Řízení projektu: Ukončení a zhodnocení iterace

Rozsah prací, které je nutno provést před zahájením další fáze projektu, závisí především na tom, jak vedoucí projektu zhodnotí možné problémy (např. pokud je budován zcela nový systém, pracovníci dodavatele mají v této oblasti málo zkušeností apod.). Je-li projekt považován za málo rizikový, stačí pro dokončení jeho úvodní fáze pouze zpřesnit některé požadavky zadání. V opačném případě je třeba naplánovat ještě další iterace, při nichž se podrobněji zanalyzují problémové oblasti, vytvoří detailnější modely případů užití a dále rozpracuje úvodní návrh architektury.

**Vstupy**

- Původní vize
- Stávající systém (pokud existuje)
- Úvodní požadavky na systém

**Výstupy**

- Vize
- Výchozí verze ekonomického plánu projektu
- Plán vývoje software - včetně odhadu potřebných zdrojů. V této fázi se nemusí být údaj ještě příliš přesný, v dalších iteracích se bude postupně zpřesňovat.

- U složitějších projektů též úvodní návrh architektury - může se jednat o "nezávazný" prototyp, který se později přepracuje.

**Milník** - cíle projektu

Na konci úvodní fáze projektu nastává důležitý milník, který rozhodne o jeho dalším pokračování. Pokud nebudou splněny následující podmínky, je třeba celý projekt přepracovat nebo v krajním případě i zastavit.

- Zadavatelé se musí shodnout s dodavateli na ceně a harmonogramu provádění prací
- Výstupní dokumenty musí prokázat, že dodavatel pochopil zadání
- Odhady rizik, nákladů a dob provádění musí být reálné

## 10.5 PŘÍPRAVA (ELABORATION)

Cílem této fáze je navrhnout stabilní model architektury systému, který poslouží jako základ pro jeho implementaci. Model architektury vychází ze základních požadavků na systém, vytváří se jeden nebo několik prototypů. Fáze přípravy navazuje na fázi zahájení projektu, na jejím začátku musí být k dispozici úvodní specifikace požadavků a úvodní verze plánu vývoje software zahrnující stručný plán iterací pro tuto fázi.

**Plán iterace** pro fázi přípravy

1. Řízení projektu: Naplánování iterace
   Plán iterace sestaví vedoucí projektu. Softwarový architekt doplní Architekturu software o popis kritérií pro její hodnocení, v případě zjištění možných problémů s architekturou aktualizuje Seznam rizik.
2. Správa prostředí: Příprava prostředí na iteraci
   V tomto stádiu je třeba nakonfigurovat vývojové nástroje a vytvořit šablony dokumentů pro analýzu a návrh (tyto činnosti zajišťuje Specialista na vývojové nástroje a Procesní inženýr).
3. Správa požadavků: Identifikace klíčových případů užití, stanovení priorit
   Softwarový architekt a vedoucí projektu určí, které případy užití se v této iteraci budou zpracovávat. Tyto případy užití budou klíčové pro návrh architektury. Vedoucí projektu následně doplní Plán iterace a provede se detailnější zpracování zvolených případů užití. Na základě nových informací pak softwarový architekt upraví model případů užití.
4. Analýza a návrh: Zpřesnění návrhu architektury
   Na základě modelu případů užití, společného slovníku a zkušeností z praxe v daném oboru navrhne softwarový architekt způsob rozdělení na subsystémy a identifikuje klíčové třídy. V této fázi se rovněž zohlední prvky stávajícího systému, které se budou s novým systémem integrovat, včetně jejich rozhraní. Návrháři poté identifikují další třídy a provedou zpřesnění jejich definice (určí jejich atributy a vzájemné vazby). Třídy, které mají vliv na architekturu systému, zapracuje systémový architekt do dokumentu "Architektura systému".
5. Analýza a návrh: Návrh subsystémů
   Softwarový architekt zpřesní definici architektury pro potřeby zvoleného implementačního prostředí (programovací jazyk, relační databáze, distribuce systému aj.) a nadefinuje subsystémy pro návrh. Tyto jsou následně přiděleny jednotlivým návrhářům.

Za použití dříve nadefinovaných instancí tříd a subsystémů popisují návrháři realizaci zvolených případů užití (návrh komponent). V tomto kroku lze též provést identifikaci klíčových prvků datového modelu.

6. Implementace: Určení fyzické distribuce systému
   *Softwarový architekt v tomto kroku navrhne počáteční strukturu modelu implementace (tj. fyzické členění systému). Vstupem tohoto postupu jsou případy užití vyjádřené pomocí diagramů spolupráce.*
7. Implementace: Plánování integrace systému
   Systémový integrátor nyní určí, v jakém pořadí se budou jednotlivé subsystémy implementovat a později integrovat v rámci prototypu architektury (je třeba aktualizovat Plán vývoje software).
8. Testování: Určení předmětu testů
   Vedoucí testování se dohodne s osobou odpovídající za příslušnou část systému, co se během této iterace bude testovat a navrhne způsob testování.
9. Implementace: Implementace komponent a Integrace subsystémů
   Implementátoři nyní provedou vlastní implementaci tříd definovaných v předchozích krocích v rámci návrhu architektury. Systém je členěn hierarchicky, jednotlivé třídy jsou logicky seskupeny do modulů a subsystémů.
   Jednotlivé části systému jsou poté samostatně odladěny a integrovány.
10. Implementace: Integrace systému
    Systémový integrátor provede postupnou integraci vytvořených subsystémů do funkčního prototypu.
11. Testování: Ověření stability systému a Testování a hodnocení
    V tomto kroku se provedou testy stávajícího systému (navrženy v kroku 8), výsledky se porovnají s požadovaným stavem. Systém testů se dle potřeby doplní pro potřeby dalšího testování.
12. Řízení projektu: Ukončení a zhodnocení iterace
    Na závěr iterace provede vedoucí projektu její hodnocení na základě srovnání plánovaných výsledků, nákladů a termínů se skutečností. V případě potřeby se v tomto kroku naplánují úpravy systému v dalších iteracích. Vedoucí projektu sestaví/upřesní plán další iterace, provede aktualizaci Plánu vývoje software a Seznamu rizik.

Výsledkem úvodní iterace by měl být první funkční prototyp architektury, jednotlivé pohledy (případy užití, logický, procesní, implementační, pohled nasazení) by měly být celkem podrobně popsány.

Další iterace v této fázi slouží rozšíření modelu návrhu a implementace o další případy užití (na základě stanovených priorit). Počet těchto iterací závisí mj. na rozsahu projektu a zkušenostech dodavatele s danou oblastí.

**Výstupy**

- Jeden či více prototypů architektury software.
- Model případů užití (nejméně z 80% kompletní). Byli identifikováni všichni aktéři a všechny klíčové případy užití, k většině případů užití existuje popis.
- Model návrhu: proveden návrh alespoň pro 10% případů užití.
- Datový model: byly identifikovány klíčové prvky datového modelu (tabulky, relace).
- Další požadavky netýkající se funkcionality a požadavky, které nelze přiřadit jednotlivým případům užití.
- Upravený seznam rizik a ekonomický plán.

- Hrubý plán projektu obsahující seznam iterací a kritéria hodnocení jednotlivých iterací.
- U systémů se složitým uživatelským rozhraním lze v této fázi vytvořit jeho prototyp.

**Milník** - architektura

Na konci fáze přípravy nastává druhý milník projektu. Projekt by v tomto stádiu měl splňovat následující podmínky:

- Vize produktu je již neměnná.
- Bylo identifikováno více než 80% všech případů užití, alespoň polovina z tohoto počtu byla již analyzována.
- Návrh a implementace byly provedeny alespoň u 10% případů užití.
- Návrh architektury je již konečný, nebude docházet k zásadním změnám.
- Jsou definovány postupy pro hodnocení a testování.
- Testování prototypů prokázalo, že byly správně identifikovány a podchyceny rizikové oblasti.
- Plán iterací pro fázi konstrukce je dostatečně podrobný a kvalitní, aby bylo možné v projektu pokračovat. Tyto plány jsou podpořeny důvěryhodnými odhady dalšího vývoje.
- Zadavatelé souhlasí s realizací software s použitím navrhované architektury a plánu vývoje; shodují se na tom, že takový software splní požadavky vytyčené ve vizi.
- Skutečná spotřeba zdrojů nepřevyšuje plán.

## 10.6 KONSTRUKCE (CONSTRUCTION)

Hlavním cílem této fáze je dokončení návrhu, vytvoření a otestování první funkční verze systému. Na rozdíl od předchozích dvou fází je tato zaměřena převážně na implementaci. Základním problémem je efektivní řízení zdrojů - stejně jako v předchozích fázích je vhodné projekt rozdělit na relativně nezávislé části a ty pak přidělit jednotlivým vývojářským týmům. V této fázi projektu by měly být požadavky na systém již stabilní, správa požadavků by měla řešit pouze případné omezuje pouze na zapracování dodatečných požadavků na změny.

**Plán iterace** pro fázi konstrukce

1. Řízení projektu: Naplánování iterace
   Vedoucí projektu naplánuje, jaká funkcionalita se během této iterace bude vytvářet.
2. Prostředí: Příprava prostředí na iteraci
   Procesní inženýr dále zpřesní Konfiguraci RUP, šablony dokumentů a pravidla vývoje.
3. Implementace: Plánování integrace systému
   Plán integrace stanovuje pořadí, ve kterém se jednotlivé moduly systému budou přidávat do testovací konfigurace. Tento plán sestavuje systémový integrátor.
4. Testování: Tvorba testů
   Návrhář testů vytvoří strukturu testů pro ověření jednotlivých požadavků na systém. Pro každý požadavek, který lze testovat, by měl být navržen alespoň jeden test (lze použít i některé upravené testy z předchozích iterací).
5. Analýza a návrh: Návrh subsystémů
   Návrháři zpřesní definici vybraných tříd/modulů určených v předchozích iteracích (např. doplní některé atributy).

6. Implementace: Implementace komponent
   Implementátoři naprogramují jednotlivé komponenty a provedou jejich otestování. Tyto základní testy by měly ověřit správnost implementace a korektní chování komponenty při vstupu správných i chybných dat.
   Během implementace se mohou objevit nové skutečnosti, na jejichž základě se provede úprava modelu návrhu.
7. Implementace: Integrace subsystémů
   Z komponent vytvořených více implementátory se na základě plánu integrace sestaví subsystém.
8. Testování: Tvorba testů
   Návrháři a implementátoři testů vytvoří testy pro ověření integrace subsystémů i celého systému. Tyto testy mají za úkol ověřit, zda jednotlivé komponenty správně spolupracují prostřednictvím použitých rozhraní.
9. Testování: Testování a hodnocení
   V tomto kroku se na příslušný subsystém aplikují testy vytvořené v krocích 4 a 6. Pokud jsou při testování zjištěny chyby, sestaví testeři hlášení o chybě (automaticky znamená i požadavek na změnu).
10. Implementace: Integrace systému
    Po sestavení a otestování subsystému je tento poskytnut integrátorovi pro začlenění do celého systému. \\ Integrátor postupně přidává jednotlivé subsystémy a sestavuje testovací verzi produktu (u počátečních verzí se testování provádí po přidání každého subsystému).
11. Testování: Testování a hodnocení
    Na celý systém se aplikují testy integrace vytvořené v kroku 6. Pokud jsou při testování zjištěny chyby, sestaví testeři hlášení o chybě.
12. Testování: Testování a hodnocení
    Cílem tohoto kroku je otestovat funkčnost celého systému. Na produkt se aplikují testy vytvořené v krocích 4 a 6. Pokud jsou při testování zjištěny chyby, sestaví testeři hlášení o chybě.
13. Řízení projektu: Ukončení a zhodnocení iterace
    Vedoucí projektu spolu se specialistou na vývojové nástroje a procesním inženýrem zhodnotí iteraci z pohledu nákladů, dodržování harmonogramu a využití vývojových nástrojů.

Další iterace ve fázi konstrukce

Další iterace slouží převážně k návrhu a implementaci další funkcionality systému. Ke konci této fáze je též kladen důraz na testování a případně i plánování nasazení systému.

**Výstupy**

- Software: První funkční verze vyvíjeného systému.
- Plán nasazení: První verze.
- Model implementace: soubor komponent vytvořených v této fázi.
- Model testování: soubor testů vyvinutý pro ověřování verzí systému vyvinutých ve fázi konstrukce.
- Uživatelská dokumentace: první verze manuálů; je třeba hlavně u systémů s rozsáhlým uživatelským rozhraním.
- Model návrhu: po dokončení této fáze by měly být identifikovány již všechny požadavky a zapracovány do modelu návrhu.

- Plán iterací pro fázi předávání.
- Datový model: úplná specifikace datového modelu (tabulky, relace, indexy…).

**Milník** - první funkční verze

Po dokončení fáze konstrukce by měl být systém připraven k předání. Kromě "alfa" či "beta" verze software musí existovat i uživatelský manuál včetně aktuální dokumentace. Projekt by v tomto okamžiku měl splňovat následující podmínky:

- Produkt je dostatečně stabilní, aby je bylo možné poskytnout uživatelům.
- Zadavatel je připraven k nasazení systému.

Pokud produkt některé tyto podmínky nesplní, je vhodné odložit předání o jednu verzi.

## 10.7 PŘEDÁVÁNÍ (TRANSITION)

Hlavním cílem této fáze je předání finální verze systému koncovým uživatelům. V jejím průběhu se provádí beta testování systému a jeho drobné úpravy na základě připomínek uživatelů. V této fázi by již nemělo docházet k žádným zásadním změnám funkcionality software - požadavky zadavatele by se měly týkat pouze instalace, konfigurace a odlaďování drobných chyb zjištěných při testovacím provozu.

**Plán iterace** pro fázi předávání

1. Řízení projektu: Naplánování iterace
   Hlavním cílem této fáze je předání software splňujícího požadavky na spolehlivost, výkon a funkcionalitu. Vedoucí projektu nyní rozhoduje o dalším vývoji hlavně na základě požadavků na změny (odstraňování závad a připomínky uživatelů k testovací verzi). Vedoucí projektu odpovídá též za plánování podpory pro koncové uživatele, zajištění materiálu potřebného pro instalaci produktu a přípravy na přejímací testy.
2. Nasazení: Beta testování
   Před vlastním oficiálním předáním produktu je vhodné dát uživatelům možnost otestovat jeho beta verzi a verzi určenou pro předání doladit na základě jejich připomínek.
3. Analýza a návrh, Správa požadavků
   V této fázi se již předpokládá, že požadavky budou téměř neměnné, nebude docházet k ad hoc úpravám produktu. Mohou se ale vyskytnout změny, které bude třeba do systému ještě zapracovat. Analýza a návrh zahrnují v této fázi hlavně drobné úpravy architektury na základě testování - ladění a změny fyzické distribuce komponent systému.
4. Implementace
   Hlavním činností v rámci implementace je odstraňování závad zjištěných při provozu beta verze sytému a zapracování případných požadavků na změny. V tomto stádiu by se již neměly vytvářet zcela nové komponenty. Stejně jako v předchozí fázi se po provedení změn provedou testy funkčnosti jednotlivých komponent.
5. Testování
   Stejně jako u předchozího kroku spočívá testování systému převážně v ověřování správnosti provedených změn. Obvykle ale není třeba zevrubně testovat integraci jednotlivých komponent, neboť jejich rozhraní jsou v tomto stádiu již stabilní. Návrh testů se omezuje na jejich přizpůsobení upraveným verzím komponent. U řady

projektů jsou též smluvně upraveny přejímací testy - jedná se o otestování předávané verze systému pomocí stávající série testů.

6. Nasazení: Přejímací testy
   Zadavatel provede před převzetím produktu jeho formální otestování. Zpravidla se použije systém testů vyvinutý v rámci testování pracovních verzí u dodavatele.
7. Výroba softwarového balíku / Poskytnutí přístupových práv ke stažení SW
   Na předání systému musí být připraveny obě smluvní strany. Dodavatel proto musí zajistit následující:
   - Vlastní dodání systému: vedoucí nasazení naplánuje vytvoření a předání instalací software (instalační programy vytvoří implementátor) a dalšího materiálu potřebného pro uvedení systému do "ostrého" provozu.
   - Vytvoření a dodání konečné verze uživatelské dokumentace: *Autor* uživatelských manuálů vytvoří jejich konečné verze.
   - Školení pracovníků zadavatele: *Autor podkladů pro školení vytvoří* poslední verze materiálů, podle nichž se bude provádět školení uživatelů.

**Výstupy**

- Software: konečná funkční verze splňující všechny požadavky zadání, jsou k dispozici instalační programy a média.
- Manuály: uživatelská dokumentace k aktuální verzi produktu.
- Materiály pro školení uživatelů k aktuální verzi produktu.

**Milník** - předání

Tímto milníkem končí vývoj stávající verze software, produkt je předán zadavateli a přechází do ostrého provozu. Základními kritérii hodnocení úspěšnosti projektu jsou spotřeba zdrojů a spokojenost uživatelů.

Pro dodavatele software začíná nyní stádium poinstalační podpory - poté, co je systém nasazen do rutinního provozu a podrobně se s ním seznámí větší množství uživatelů, obdrží zadavatel zpravidla řadu návrhů na jeho zlepšení.

V závislosti na závažnosti připomínek se nyní rozhodne, zda se zahájí nový projekt a vytvoří další verze celého software či zda bude pouze stačit provést úpravy některých komponent.

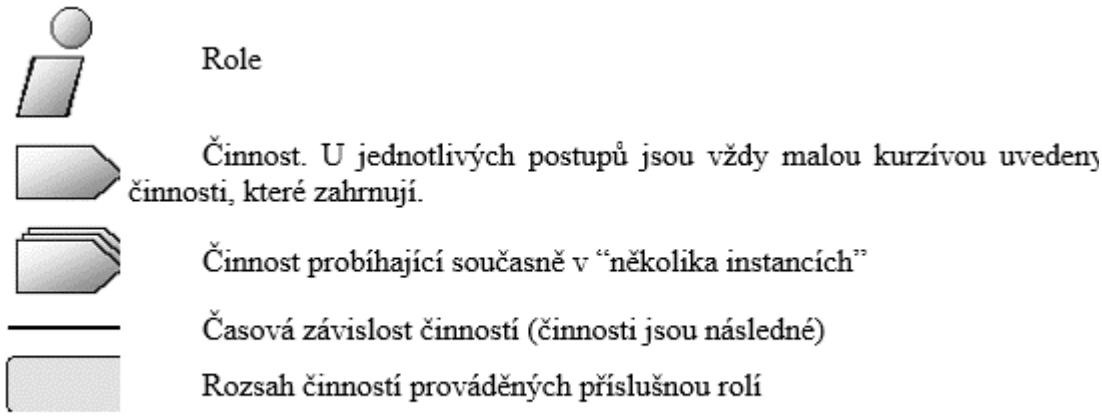
## 10.8 DISCIPLINY SPOJENÉ S PROJEKTEM

Přestože názvy a pořadí prvních šesti disciplin mohou vyvolávat určité asociace s "vodopádovým" přístupem k vývoji software, je třeba mít na paměti určitou odlišnost: v metodice RUP se tato posloupnost v průběhu projektu aplikuje vícekrát, podíl jednotlivých činností však závisí na stádiu, v němž se produkt nachází.

### *10.8.1 VYSVĚTLENÍ POUŽITÝCH SYMBOLŮ*

Ve schématech jednotlivých postupů byly použity následující symboly (shodně s [RUP]):

**Obrázek 26:** Symboly používané v RUP




Zdroj: ALDORF F., Metodika RUP

### *10.8.2 DISCIPLINY V RUP*

RUP rozlišuje mezi tzv. "hlavními" a "podpůrnými" disciplinami. Hlavní discipliny zajišťují vlastní práci na projektu, podpůrné discipliny slouží k jejich řízení a koordinaci. Jedná se o následující:

#### *Hlavní discipliny*

- Tvorba podnikového modelu
- Správa požadavků
- Analýza a návrh
- Implementace
- Testování
- Nasazení

### *Podpůrné disciplíny*

- Řízení projektu
- Řízení změn a konfigurace
- Správa prostředí

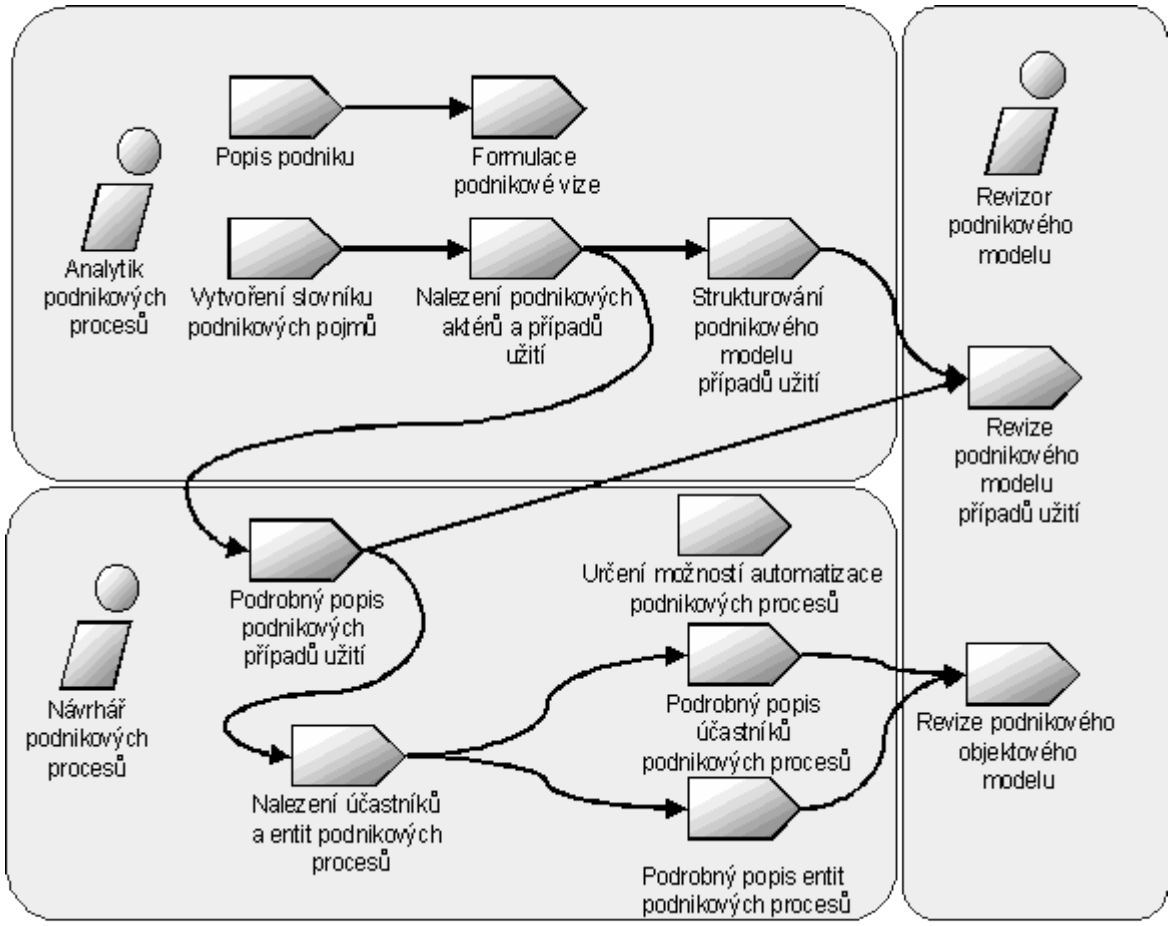
### *10.8.3 TVORBA PODNIKOVÉHO MODELU*

Problémem řady projektů je špatná komunikace pracovníků odpovědných za modelování podnikových procesů a softwarovými návrháři. Metodika Rational Unified Process propojuje obě činnosti na úrovni řízení projektu, pro lepší komunikaci se využívá společná terminologie (významové slovníky).

Pro tvorbu podnikového modelu se využívají prostředky UML: jednotlivé podnikové scénáře jsou znázorněny pomocí podnikového modelu případů užití, jejich realizaci popisuje podnikový objektový model.

U systémů, které nejsou svázány s konkrétním podnikovým procesem, se tato činnost zpravidla vypouští.

**Obrázek 27:** Tvorba podnikového modelu




Zdroj: ALDORF F., Metodika RUP

### *Postupy v rámci tvorby podnikového modelu*

- **Základní charakteristika podniku:** Analytik podnikových procesů na základě konzultací se zadavatelem sepíše základní charakteristiky podniku a formuluje jeho vizi.
    - Popis podniku
    - Formulace podnikové vize
    - Tvorba slovníku podnikových pojmů
- **Nalezení podnikových procesů:** Cílem tohoto kroku je identifikovat podnikové procesy (pomocí modelu případů užití) a určit, na které se další vývoj systému zaměří. Pro sjednocení terminologie se průběžně doplňuje slovník podnikových pojmů.
    - Tvorba slovníku podnikových pojmů
    - Nalezení podnikových aktérů a případů užití
- **Podrobný popis podnikových procesů:** Procesy identifikované v předchozím kroku jsou přiděleny týmům analytiků, kteří je podrobně popíší pomocí strukturovaných modelů případů užití. Návrhář podnikových procesů provede detailní popis jednotlivých případů užití.
  Správnost výsledného modelu pak ověří návrhář podnikových procesů ve spolupráci se zadavatelem (tj. tým v roli revizora podnikového modelu).
    - Strukturování podnikového modelu případů užití
    - Podrobný popis podnikových případů užití
    - Revize podnikových modelů případů užití
- **Popis realizace podnikových procesů:** Poté, co byly jednotlivé procesy vymezeny a podrobně popsány pomocí modelu případů užití, je možné přistoupit k popisu jejich realizace (nalézt jejich účastníky, vstupy, výstupy a určit vazby mezi nimi). Výsledkem tohoto kroku je podnikový objektový model.
    - Nalezení účastníků a entit podnikových procesů
    - Tvorba podnikového objektového modelu
- **Zpřesnění podnikového objektového modelu:** V objektovém modelu vytvořeném v předchozím kroku se doplní specifikace jednotlivých účastníků (tj. jejich odpovědnosti) a entit. Poté se ve spolupráci se zadavatelem provede revize podnikového objektového modelu.
    - Podrobný popis entit podnikových procesů
    - Podrobný popis účastníků podnikových procesů
    - Revize podnikového objektového modelu
- **Určení možností automatizace procesů:** Na základě popisu realizace podnikových procesů se určí možnosti jejich realizace a z nich vyplývající požadavky na systém.
    - Určení možností automatizace podnikových procesů

### *10.8.4 SPRÁVA POŽADAVKŮ*

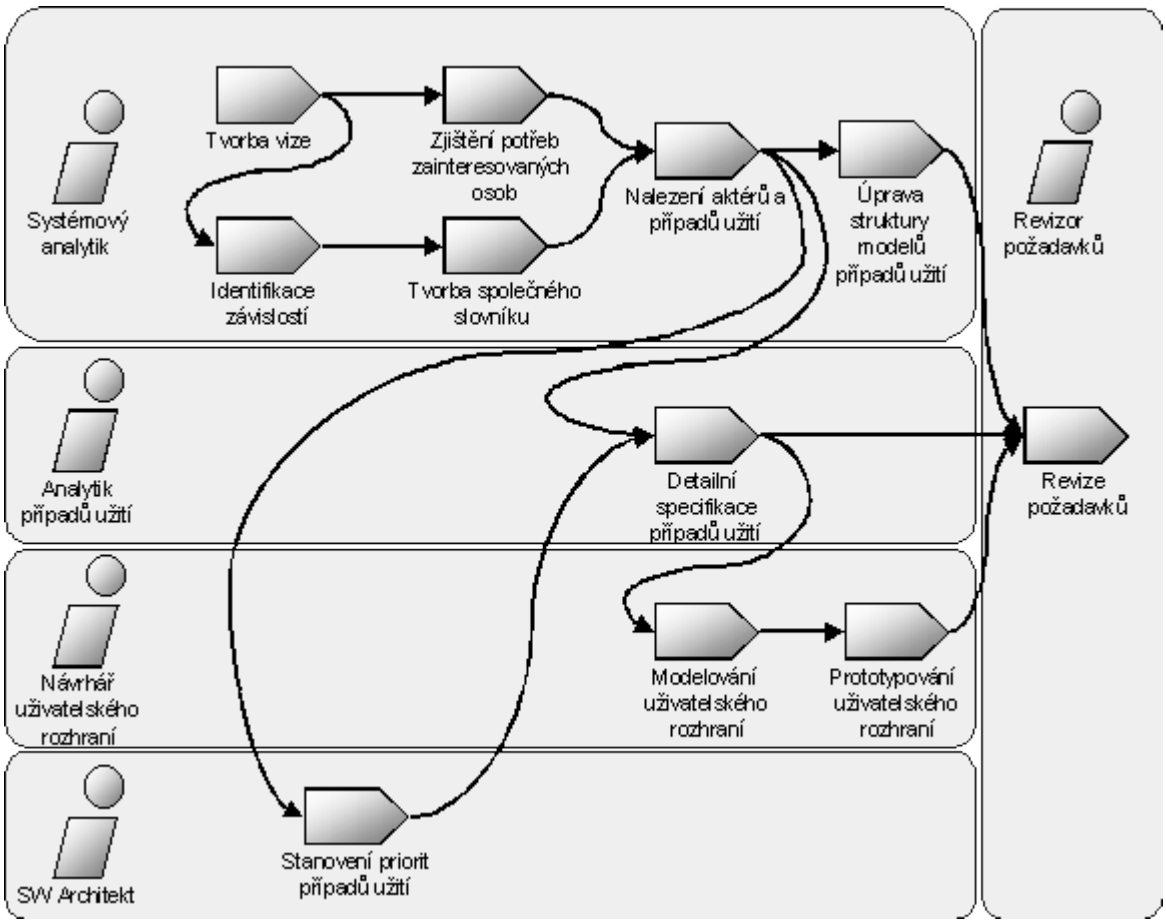
Požadavky jsou kritérii, jejichž splnění podmiňuje úspěch projektu. Požadavky je proto třeba systematicky evidovat a odpovídajícím způsobem dokumentovat a zapracovávat jejich případné změny.

Správu požadavků bychom tedy mohli definovat jako (blíže viz [UC]):

- systematický přístup ke zjišťování a dokumentaci požadavků na systém a
- proces zajišťující shodu mezi zadavatelem a dodavatelem v případě změny požadavků

na systém.

**Obrázek 28:** Správa požadavků




Zdroj: ALDORF F., Metodika RUP

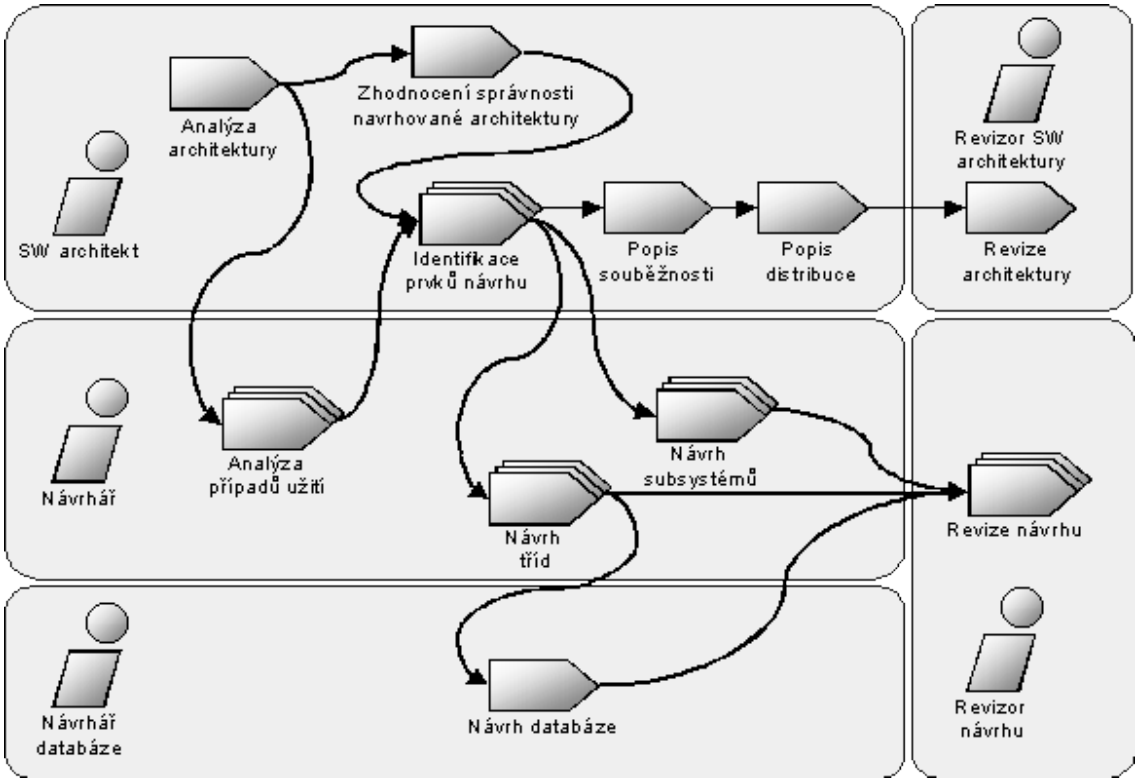
- **Analýza problémové oblasti:** Klíčovou činností v rámci analýzy problémové oblasti je tvorba vize. Tuto činnost provádí systémový analytik na základě konzultací se zainteresovanou osobou (většinou z řad zadavatele). Jejím cílem je příslušný problém přesně identifikovat a popsat, určit další osoby, kterých se dotýká a stanovit, co všechno by úspěšné řešení mělo splňovat.
  V průběhu správy požadavků se vytváří společný slovník, který má za úkol sjednotit používanou terminologii a pomoci odstranit možné duplicity způsobené rozdílnou formulací téhož problému.
  V rámci tohoto kroku se též identifikují základní závislosti mezi jednotlivými požadavky a klíčoví aktéři systému (tj. osoby či jiné systémy, které systém využívají).
  - Zjištění potřeb zainteresovaných osob
  - Tvorba vize
  - Tvorba společného slovníku
  - Identifikace závislostí
  - Nalezení aktérů a případů užití
- **Identifikace požadavků zadavatele:** Prokáže-li úvodní verze vize, že se vyplatí v projektu pokračovat, provede se "sběr" požadavků zainteresovaných osob (např. formou elektronických příspěvkových skupin). Systémový analytik tyto informace utřídí, odstraní z nich duplicity a na jejich základě doplní vizi a určí základní případy užití systému. Výsledkem by měl být úvodní model případů užití obsahující klíčové

prvky systému.
    - Zjištění potřeb zainteresovaných osob
    - Tvorba vize
    - Tvorba společného slovníku
    - Identifikace závislostí
    - Nalezení aktérů a případů užití
- **Definice systému:** Tento krok představuje další zpřesnění modelu případů užití, jeho výsledkem by měla být kompletní specifikace požadavků na úrovni systému.
    - Tvorba společného slovníku
    - Identifikace závislostí
    - Nalezení aktérů a případů užití
- **Identifikace klíčových případů užití, stanovení priorit:** Poté, co byli určeni téměř všichni aktéři i případy užití systému, lze přistoupit ke stanovení jejich priorit i možných rizik. Softwarový architekt v tomto kroku nalezne případy užití klíčové pro architekturu systému a vytvoří její první nástin (obsahující pouze pohled případů užití). Priority případů užití se poté zanesou do vize.
    - Tvorba vize
    - Stanovení priorit případů užití
    - Identifikace závislostí
- **Zpřesnění definice systému:** Tento krok předpokládá, že na úrovni systému byli identifikováni všichni a aktéři i případy užití a ke všem existuje alespoň stručný popis. Analytik případů užití detailně popíše jednotlivé případy užití (vstupní i výstupní podmínky, postupy v rámci případu užití).
  U systémů, jejichž důležitou částí je uživatelské rozhraní, se na základě přesné specifikace případů užití provede jeho návrh.
    - Detailní specifikace případů užití
    - Modelování uživatelského rozhraní
    - Prototypování uživatelského rozhraní
- **Řízení změn požadavků:** V průběhu projektu podávají jednotliví účastníci požadavky na změny. Systémový analytik je nejprve klasifikuje a určí, zda se jedná o požadavky na odstranění chyb software, změnu stávajícího požadavku nebo přidání nové funkcionality.
  Požadavky se pak popíší pomocí výše uvedených postupů, předají se ke zhodnocení revizorovi požadavků (pracovníci odpovídající za části systému, kterých se změna dotkne). V případě, že požadavky jsou relevantní a z hlediska časového plánu a rozpočtu proveditelné, zajistí jejich zapracování do projektu.
    - Identifikace závislostí
    - Úprava struktury požadavků
    - Revize požadavků

### *10.8.5 ANALÝZA A NÁVRH*

Analýza a návrh propojuje požadavky na systém s jeho implementací. Zatímco modely případů užití vytvářené v rámci správy požadavků odpovídá na otázku “co se bude vytvářet?”, návrh by měl řešit “jak se to bude vytvářet?” - tj. nadefinovat architekturu systému včetně jeho členění na komponenty.

**Obrázek 29:** Revize návrhu a architektury




Zdroj: ALDORF F., Metodika RUP

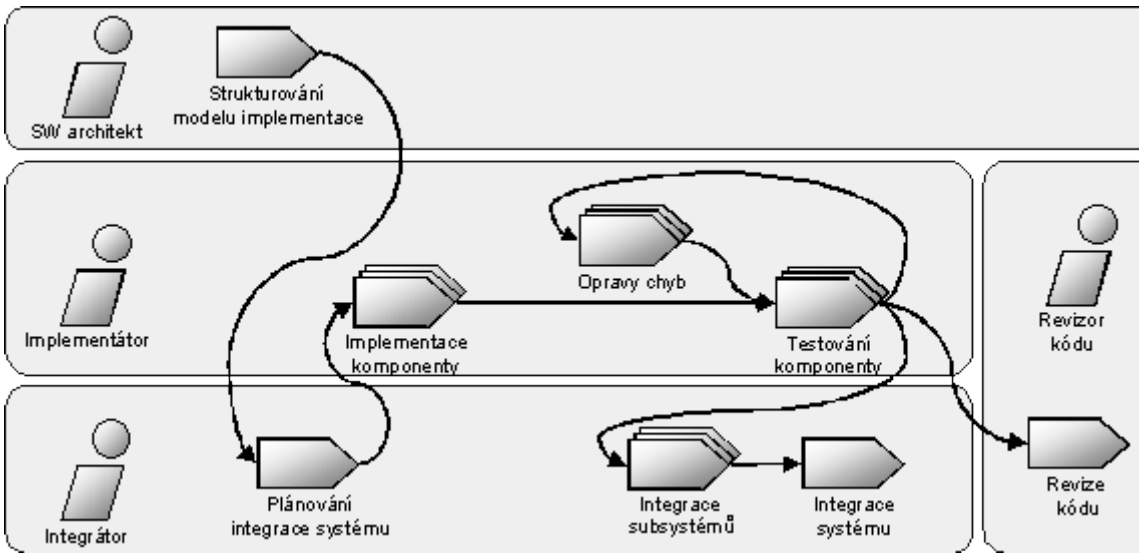
- **Úvodní návrh architektury:** Cílem tohoto kroku je na základě analýzy případů užití a dalších požadavků nalézt klíčové třídy a vytvořit analytický model systému. V tomto stádiu by se ale spíše než o komplexní návrh systému mělo jednat o nástin řešení, na jehož základě bude možné posoudit jeho proveditelnost.
  - Analýza architektury
  - Analýza případů užití
  - Zhodnocení správnosti navrhované architektury
- **Zpřesnění návrhu architektury:** Tento krok představuje přechod od analýzy k návrhu. Analytický model je doplněn o aspekty týkající se implementace a nasazení systému - SW architekt nyní na základě analýzy případů užití identifikuje jednotlivé prvky návrhu (tj. budoucí komponenty). V tomto stádiu se též řeší souběžnost procesů a fyzická distribuce systému.
  - Identifikace prvků návrhu
  - Popis souběžnosti
  - Popis distribuce
- **Návrh subsystémů:** Poté, co je k dispozici definice základních prvků návrhu, vytvoří k nim návrhář přesnější popis jejich funkcionality, aby je by bylo možné implementovat jako komponenty. Nyní se též navrhuje hierarchická struktura modelu implementace.
  - Návrh tříd
  - Návrh subsystémů

- Revize návrhu
- **Návrh databáze:** Návrhář databáze určí perzistentní třídy a navrhne k nim příslušné datové struktury.
  - Návrh databáze
  -

### *10.8.6 IMPLEMENTACE*

Implementace má za úkol komponenty vymezené v modelu návrhu převést do spustitelné podoby - tj. vytvořit programy, knihovny a všechny potřebné datové soubory. V rámci implementace se též provádí integrace komponent a základní testy jejich funkčnosti.

**Obrázek 30:** Implementace




**Zdroj: ALDORF F., Metodika RUP**

- **Určení fyzické distribuce systému:** Softwarový architekt na základě modelu návrhu
- vymezí způsob implementace jednotlivých komponent systému. Tento pracovník by měl mít určité zkušenosti s používaným implementačním prostředím a integrací, neboť nesprávně zvolená struktura systému mj. podstatně ztíží souběžný vývoj komponent systému.

  - Strukturování modelu implementace

- **Plánování integrace systému:** Cílem tohoto postupu je naplánovat, které subsystémy se budou v rámci aktuální iterace implementovat a v jakém pořadí se provede jejich integrace.

  - Plánování integrace systému

- **Implementace komponent:** Implementátoři vytvoří zdrojový kód komponent a zkompilují. Poté jej otestují a odstraní případné chyby.
  Revizor kódu pak zkontroluje, zda zdrojový kód splňuje formální požadavky dané pravidly programování.

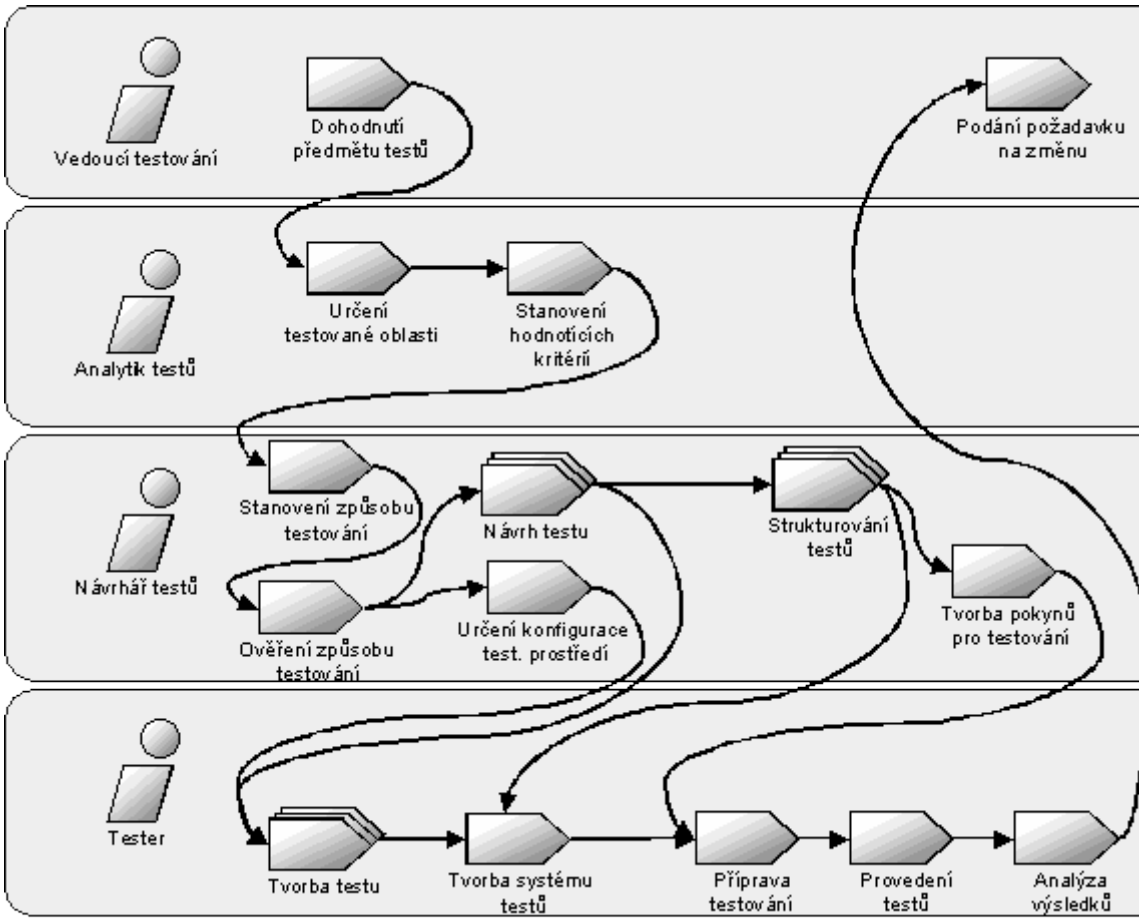
- Implementace komponenty
- Testování komponenty
- Opravy chyb
- Revize kódu

- **Integrace subsystémů:** V případě, že subsystém je vyvíjen týmově, zajistí pověřený člen týmu integraci komponent. Poté, co je celý subsystém otestován (viz následující podkapitola), je zařazen do repozitáře pro integraci.
  - Integrace subsystému
- **Integrace systému:** Integrátor postupně přidává jednotlivé subsystémy dle plánu integrace, v každém kroku je dílčí systém otestován (viz následující podkapitola).
  - Integrace systému

### *10.8.7 TESTOVÁNÍ*

Cílem testování je vytvořit a provést soubor testů pro ověření funkčnosti komponent systému a jejich správné integrace.

**Obrázek 31:** Testování




Zdroj: ALDORF F., Metodika RUP

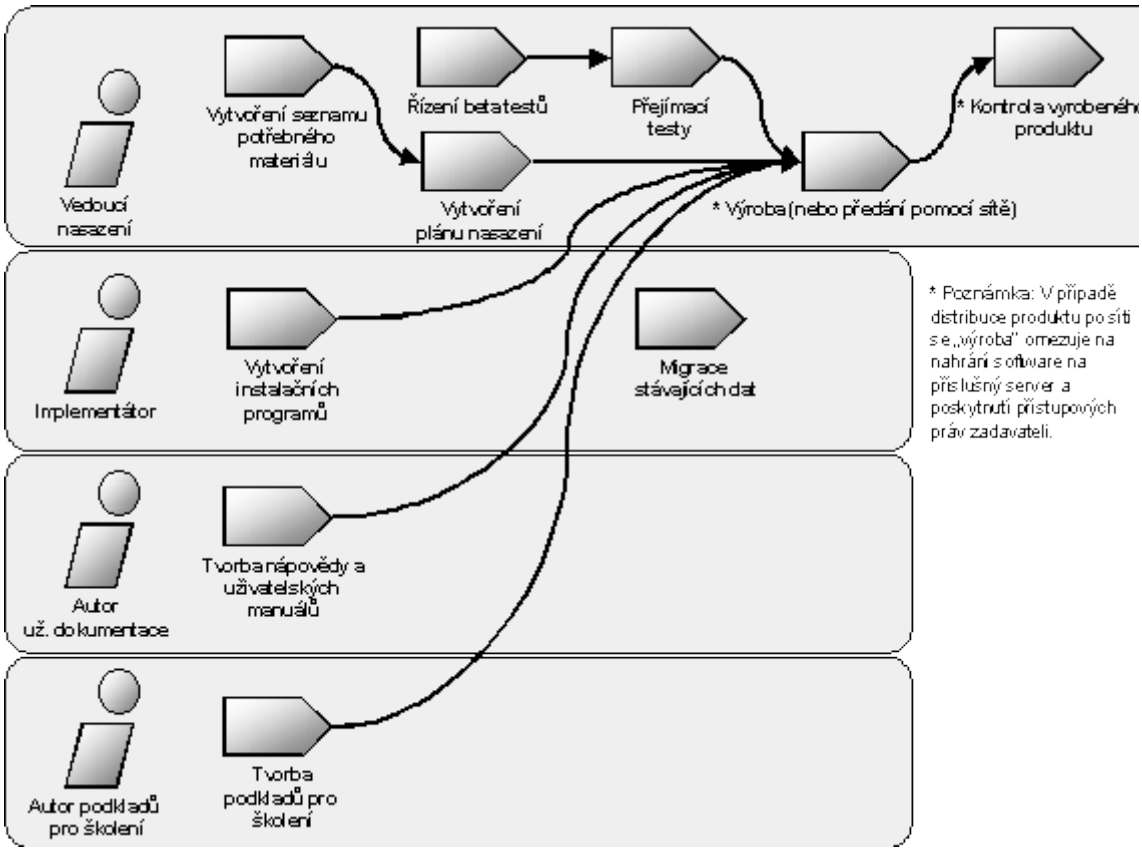
- **Určení předmětu testů:** Tento postup se aplikuje v rámci plánování každé iterace. Jeho cílem je ve spolupráci s osobou odpovědnou za příslušnou část systému stanovit co a jak se během dané iterace bude testovat.

    - Dohodnutí předmětu testů
    - Určení testované oblast
    - Stanovení hodnotících kritérií
    - Určení způsobu testování
- **Ověření způsobu testování:** Tento postup má za úkol prakticky ověřit, že způsob testování zvolený v předchozím kroku je vhodný.
    - Určení konfigurace testovacího prostředí
    - Tvorba testu
    - Ověření způsobu testování
- **Ověření stability systému:** Cílem tohoto postupu je ověřit, zda je systém již dostatečně stabilní a provedení komplexních testů má smysl. Stabilita systému se zpravidla testuje pomocí automatizovaných testovacích procedur.
    - Návrh testu
    - Tvorba testu
    - Provedení testů
    - Analýza výsledků
- **Tvorba testů:** Vlastní implementace testů pro danou iteraci.
    - Návrh testu
    - Tvorba testu
    - Tvorba systému testů
- **Testování a hodnocení:** Pokud jsou při testech zjištěny jakékoli nedostatky, jsou zdokumentovány a dále zpracovány jako požadavky na změnu.
    - Příprava testování
    - Provedení testů
    - Analýza výsledků

### *10.8.8 NASAZENÍ*

Nasazení systému je hlavní činností prováděnou ve fázi předávání, již před jejím zahájením by však mělo být důkladně naplánováno. Jeho hlavním cílem je poskytnutí software koncovým uživatelům.

**Obrázek 32:** Nasazení




Zdroj: ALDORF F., Metodika RUP

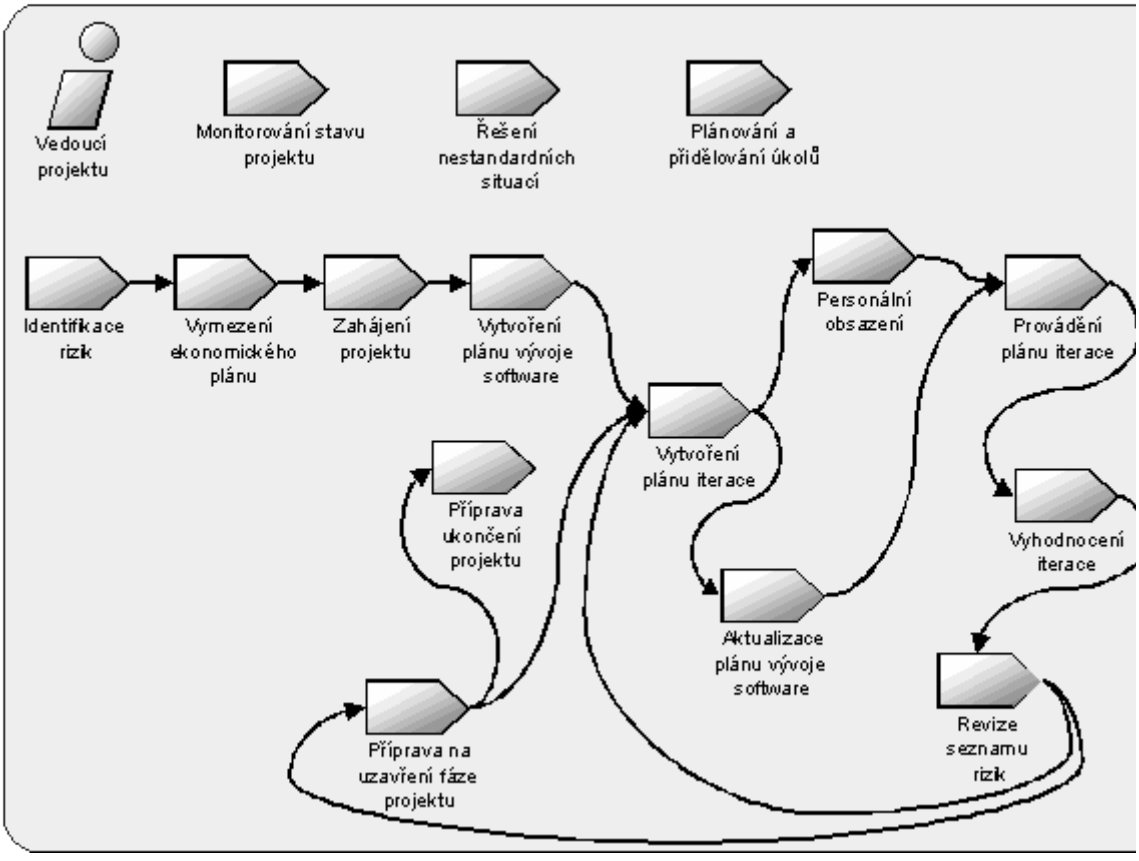
- **Plánování nasazení:** Před vlastním nasazením software je třeba určit jeho způsob (zda bude distribuován na médiích nebo po síti) a vytvořit jeho časový plán. Vzhledem k tomu, že kromě software se zpravidla dodávají i další položky (manuály, hardwarové komponenty aj.), je třeba tyto rovněž dokumentovat a zohlednit v plánu.
    - Vytvoření seznamu potřebného materiálu
    - Vytvoření plánu nasazení
- **Tvorba dokumentace:** Aby pracovníci zadavatele mohli produkt samostatně používat, je třeba jim poskytnout dokumentaci v podobě uživatelských manuálů. U některých produktů se též vytváří materiály pro školení uživatelů.
    - Tvorba nápovědy a uživatelských manuálů
    - Tvorba podkladů pro školení
- **Vytvoření instalace:** Cílem tohoto postupu je vytvoření instalace produktu určenou koncovým uživatelům. V závislosti na stádiu vývoje se může jednat o testovací (beta) či konečnou verzi.
    - Vytvoření instalačních programů
- **Beta testování:** Funkční verze programu je předána určité skupině koncových uživatelů, aby provedli její testování při běžném provozu. Zjištěné závady jsou zaevidovány a v "ostré" verzi produktu odstraněny.
    - Řízení beta testů
- **Přejímací testy:** Před vlastním předáním produktu se obvykle provedou testy, v nichž se zadavatel přesvědčí o jeho správné funkčnosti. Testování se zpravidla provádí již v

prostředí, do kterého se bude produkt nasazovat - tj. v systému zadavatele.
    - Řízení přejímacích testů
- **Výroba softwarového balíku:** Tento krok se použije v případě, že produkt je distribuován jako krabicový software.
  Pokud jsou přejímací testy úspěšné, předá vedoucí nasazení produkt do výroby a provede jeho ověření.
    - Výroba
    - Kontrola vyrobeného produktu
- **Poskytnutí přístupových práv ke stažení SW:** Tento krok se použije v případě, že

### *10.8.9 ŘÍZENÍ PROJEKTU*

Řízení projektu je hlavní podpůrnou disciplínou zajišťující plánování a koordinaci prací na projektu. Významným úkolem této disciplíny je též zajištění potřebných personálních i materiálních zdrojů.

**Obrázek 33:** Řízení projektu




Zdroj: ALDORF F., Metodika RUP

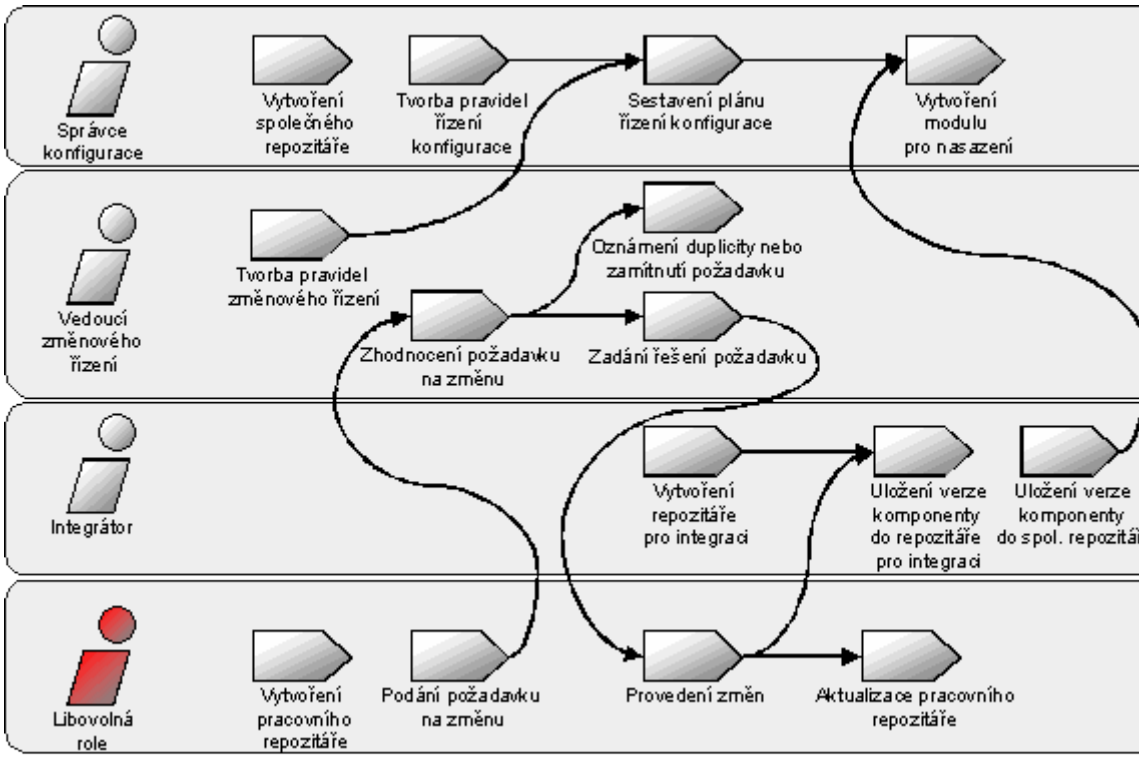
- **Zahájení projektu:** Na základě zadání se vytvoří úvodní seznam rizik spojených s projektem a provede se ekonomická analýza, jejímž výstupem je ekonomický plán. Na základě těchto podkladů se vedoucí projektu rozhodne, zda se software vyplatí vyvíjet či nikoliv.
    - identifikace rizik

    - vymezení ekonomického plánu
    - zahájení projektu
- **Plánování vývoje software:** V tomto kroku se vytvoří či aktualizuje plán vývoje software.
    - vytvoření plánu vývoje software
- **Naplánování iterace:** Před zahájením každí iterace je třeba vytvořit podrobný plán činností, využití zdrojů a kritérií pro její zhodnocení. K tomu slouží plán iterace.
    - vytvoření plánu iterace
    - aktualizace plánu vývoje software
- **Řízení iterace:** Tento postup je realizací plánu vytvořeného v předchozím kroku. Každá iterace začíná zajištěním potřebných personálních i materiálních zdrojů a končí zhodnocením a porovnáním výsledků s plánem.
    - Personální obsazení
    - Provádění plánu iterace
- **Ukončení a zhodnocení iterace:** Na závěr každé iterace se provede její zhodnocení z pohledu dodržení harmonogramu a plánované spotřeby zdrojů. V případě potřeby se provede aktualizace seznamu rizik.
    - Vyhodnocení iterace
    - Revize seznamu rizik
- **Monitorování a řízení projektu:** Jedná se o soubor činností, které vedoucí projektu vykonává po celou dobu trvání projektu.
    - Monitorování projektu
    - Řešení nestandardních situací
    - Plánování a přidělování úkolů
- **Ukončení fáze projektu:** Pokud projekt dospěje ke konci fáze (splňuje hodnotící kritéria pro příslušný milník), provede vedoucí projektu formální ukončení fáze.
    - příprava na uzavření fáze projektu
- **Ukončení projektu:** Software se předá zadavateli a provede se zhodnocení celého projektu.
    - příprava ukončení projektu

### *10.8.10 Řízení změn a konfigurace*

Tato disciplina je nezbytná pro zachování konzistence systému, neboť jejím hlavním úkolem je řídit verzování software, vytvořit systém předávání posledních verzí veškerých výstupů a koordinovat implementaci změn.

**Obrázek 34:** Řízení změn a konfigurace




Zdroj: ALDORF F., Metodika RUP

### Postupy v rámci řízení změn a konfigurace

- **Plánování konfigurace projektu a řízení změn:**
  - tvorba plánu konfigurace
  - tvorba pravidel řízení konfigurace
  - tvorba pravidel změnového řízení
- **Vytvoření prostředí pro řízení konfigurace:** Správce konfigurace spolu se softwarovým architektem vytvoří společný repozitář podle struktury komponent produktu. Tento repozitář bude sloužit ke sdílení dat a kódu při vývoji software. Integrátor vytvoří repozitář pro integraci, do něhož se budou předávat hotové a řádně otestované komponenty ze společného repozitáře (z nich se pak sestaví verze software pro testování integrace a předání).
  - vytvoření společného repozitáře
  - vytvoření repozitáře pro integraci
- **Změna a předávání částí konfigurace:** Jedná se o postup vymezující přístup pracovníka v libovolné roli k projektovým datům, provádění a publikaci změn.
  - vytvoření pracovního repozitáře
  - provedení změn
  - aktualizace pracovního repozitáře
  - uložení verze komponenty do repozitáře pro integraci
- **Správa verzí:** Pokud pracovní verze určitého subsystému dospěje do použitelného stádia, uloží ji integrátor do společného repozitáře pro sestavení celého produktu či využití v dalších iteracích.
  - Uložení verze komponenty do společného repozitáře

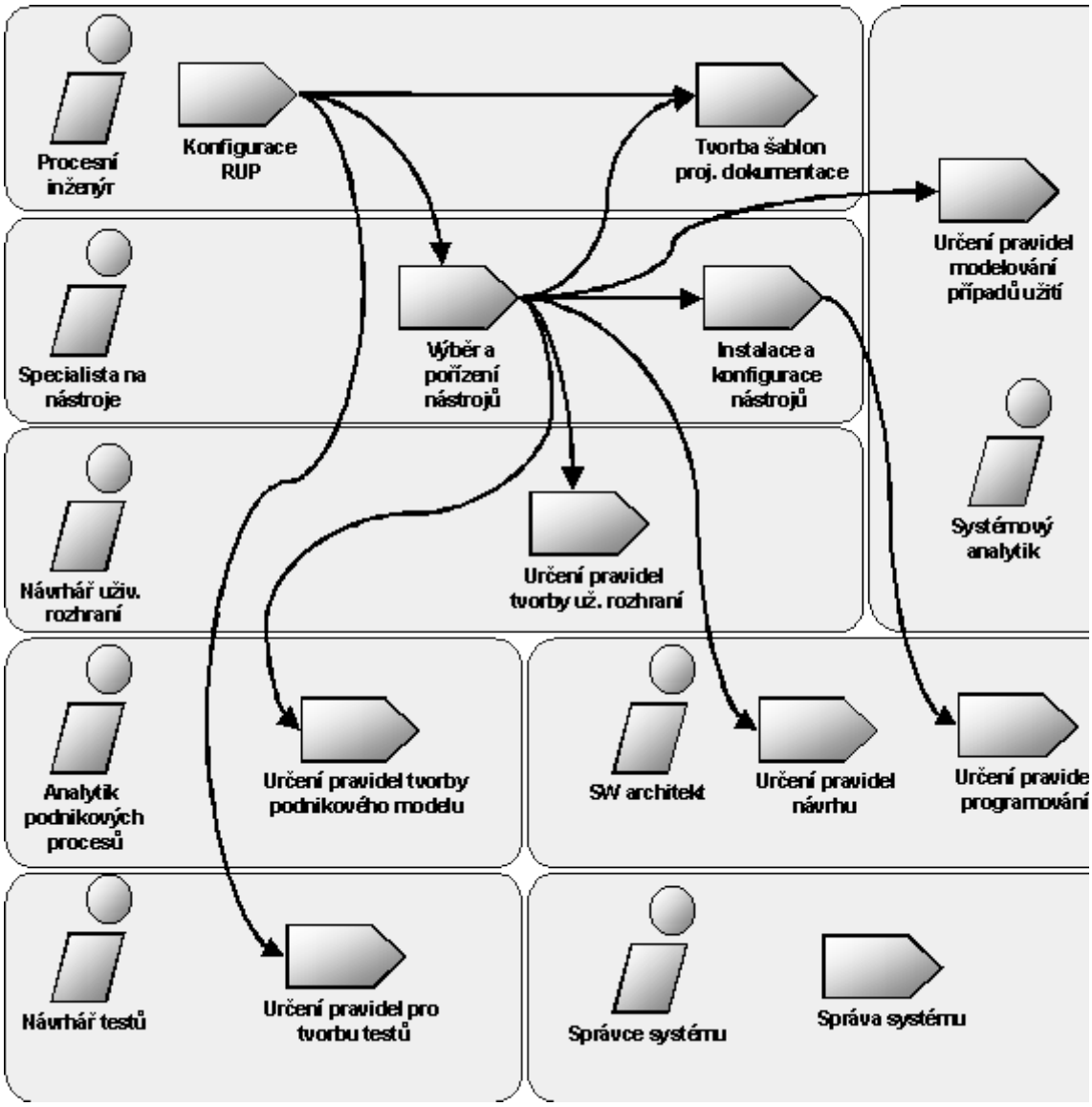
    - Vytvoření modulu pro nasazení
- **Řízení změn:** Standardizovaný, centrálně řízený postup řízení a dokumentace změn umožňuje udržení systému v konzistentním stavu i v případě úprav.
    - Podání požadavku na změnu
    - Zhodnocení požadavku na změnu
    - Oznámení duplicity nebo zamítnutí požadavku
    - Zadání řešení požadavku

### *10.8.11 Správa prostředí*

Zatímco řízení projektu řeší otázky plánování a koordinace - tj. "kdo kdy a co bude dělat", úkolem správy prostředí je určit "jak" (stanovit pravidla pro jednotlivé činnosti) a "čím" (dát k dispozici příslušné nástroje).

- **Příprava prostředí projektu:** Cílem tohoto kroku je příprava organizace dodavatele na zahájení projektu. Procesní inženýr vytvoří šablony klíčových projektových dokumentů a Konfiguraci RUP. Specialista na vývojové nástroje zvolí a pořídí příslušné prostředky pro podporu vývoje.
    - Konfigurace RUP
    - Tvorba šablon projektové dokumentace
    - Výběr a pořízení vývojových nástrojů
    -
- **Příprava prostředí na iteraci:** Před zahájením iterace musí existovat popis vytvářených dokumentů (tj. je třeba doplnit Konfiguraci RUP) a měly by být k dispozici šablony pro jejich tvorbu. Jsou-li pro iteraci potřeba nějaké softwarové nástroje, zajistí specialista na vývojové nástroje jejich instalaci a konfiguraci.
    - Konfigurace RUP
    - Tvorba šablon proj. dokumentace
    - Instalace a konfigurace nástrojů

**Obrázek 35:** Správa prostředí




Zdroj: ALDORF F., Metodika RUP

**Příprava pravidel pro iteraci:** Cílem tohoto postupu je vytvořit jednotná pravidla pro jednotlivé činnosti prováděné během dané iterace. To, které z níže uvedených

- činností se budou provádět, záleží samozřejmě na stádiu projektu.
    - Určení pravidel pro tvorbu už. rozhraní
    - Určení pravidel tvorby podnikového modelu
    - Určení pravidel pro tvorbu testů
    - Určení pravidel modelování případů užití
    - Určení pravidel návrhu
    - Určení pravidel programování
- **Podpora prostředí během iterace:** Správce systému zajišťuje v průběhu celého projektu funkčnost vývojových nástrojů a řeší případné problémy vývojářů.
    - Správa systému

## 10.9 OSOBY PODÍLEJÍCÍ SE NA PROJEKTU (ROLE)

Tato podkapitola obsahuje přehled osob podílejících se na vývoji software. U některých (zvláště malých) projektů nemusí být některé zde uvedené role vůbec zastoupeny, nebo že jeden pracovník současně působí v několika rolích. Není též neobvyklé, že jedna role je zastoupena větším počtem osob (typické např. u implementátorů).

Seznam je pro větší přehlednost členěn podle činností, na nichž se příslušní pracovníci nejvíce podílejí.

### *10.9.1 ŘÍZENÍ PROJEKTU*

- **Zadavatel:** Osoba/organizace zainteresovaná na tvorbě systému. Zadavatel definuje požadavky na systém a financuje projekt.
- **Vedoucí projektu:** Osoba odpovídající za projekt jako celek. Vedoucí projektu má za úkol řídit vývojářský tým tak, aby výsledný produkt splňoval požadavky zadavatele na kvalitu i funkcionalitu a jeho vývoj probíhal podle dohodnutého harmonogramu.

### *10.9.2 TVORBA PODNIKOVÉHO MODELU*

- **Analytik podnikových procesů:** Úkolem této role je popsat fungování procesů na úrovni případů užití (tj. identifikovat jednotlivé aktéry a případy užití, vymezit vztahy mezi nimi a výsledný model strukturovat).
- **Návrhář podnikových procesů:** Zajišťuje popis realizace jednotlivých podnikových případů užití: vyhledá jednotlivé entity a účastníky procesů, vytváří podnikový objektový model; v rámci případů užití modeluje postupy a určuje možnosti jejich automatizace.
- **Revizor podnikového modelu:** Na rozdíl od jednotlivých návrhářů podnikových procesů má obecné znalosti o všech modelovaných procesech a jejich celkovém kontextu. Jeho úkolem je proto ověřit správnost podnikového modelu jako celku.

### *10.9.3 SPRÁVA POŽADAVKŮ*

- **Systémový analytik:** Jeho hlavním úkolem je zjistit požadavky zadavatele na systém a systemizovat je pomocí modelů případů užití (či slovních popisů). Pro potřeby sjednocení terminologie vytváří společný slovník.
- **Návrhář uživatelského rozhraní:** Vytváří prototypy uživatelského rozhraní, na základě připomínek uživatelů a implementátorů tyto doplňuje a upravuje. Ve spolupráci s implementátory a uživateli vytváří uživatelské rozhraní systému.

### *10.9.4 SPRÁVA PROSTŘEDÍ*

- **Procesní inženýr:** Odpovídá za vlastní proces vývoje. Vzhledem k tomu, že RUP je univerzální, velmi komplexní metodika, je třeba ji pro potřeby konkrétního projektu "konfigurovat". Procesní inženýr řeší, jak se jednotlivé činnosti budou provádět a dokumentovat.
- **Specialista na vývojové nástroje:** Úkolem této role je vybrat, pořídit a

nakonfigurovat vhodné nástroje pro podporu vývoje software.
- **Správce systému:** Zajišťuje správu a údržbu systému, na kterém se projekt zpracovává. Správce systému má na starosti činnosti jako správu uživatelských kont, opravy hardware a zálohování.

### *10.9.5 ANALÝZA A NÁVRH*

- **Softwarový architekt:** V průběhu celého projektu řídí a koordinuje všechny činnosti související s technologií. Jeho úkolem je nadefinovat strukturu jednotlivých pohledů v rámci architektury (tj. dekompozici v rámci příslušného pohledu a vymezení rozhraní jednotlivých částí). Architekt by měl mít podobnou kvalifikaci jako návrhář, na rozdíl od něj ovšem musí mít přehled o celém systému bez potřeby detailních znalostí jednotlivých částí.
- **Návrhář:** Odpovídá za popis případů užití systému pomocí modelu tříd, vymezení jednotlivých postupů pomocí dynamických modelů a určení způsobu jejich implementace.
- **Návrhář databáze:** Řeší problematiku ukládání dat a práce s perzistentními objekty. Navrhuje databáze, strukturu tabulek, způsob indexování aj. Výstupem jeho činnosti je datový model.
- **Revizor architektury:** Kontroluje správnost návrhu architektury systému.
- **Revizor návrhu:** Má za úkol ověřit celkovou správnost modelu návrhu. Na rozdíl od návrhářů, kteří musejí dokonale znát pouze určitou část systému, musí revizor mít revizor návrhu přehled o celém systému; příliš detailní znalosti nejsou třeba.

### *10.9.6 TESTOVÁNÍ*

- **Vedoucí testování:** Odpovídá za plánování a koordinaci testů.
- **Analytik testů:** Má za úkol určit, jaká část aplikace se bude testovat, a stanovit hodnotící kritéria.
- **Návrhář testů:** Navrhuje a ověřuje způsob testování.
- **Tester:** Odpovídá za vlastní implementaci a provedení testů, zaznamenává a vede evidenci výsledků jednotlivých testů, v případě problémů informuje vedoucího testování.

### *10.9.7 IMPLEMENTACE*

- **Implementátor:** Odpovídá za vytvoření kódu jednotlivých komponent a subsystémů. Implementátor provádí též základní testování vytvořených komponent a opravuje případné chyby.
- **Systémový integrátor:** Má za úkol z komponent dodaných implementátory vytvořit postupně subsystémy a výsledný systém (otestované komponenty se předávají z pracovního repozitáře jednotlivých implementátorů do repozitáře pro integraci). Integrátor odpovídá též za plánování integrace.
- **Revizor kódu:** Provádí kontrolu zdrojového kódu a zadává řešení případných nedostatků.

### 10.9.8 NASAZENÍ

- **Vedoucí nasazení:** Plánuje a řídí proces předávání produktu do provozu.
- **Autor podkladů pro školení:** Vytváří elektronické prezentace a jiné materiály sloužící pro zaškolování uživatelů.
- **Autor uživatelské dokumentace:** Vytváří uživatelskou dokumentaci produktu: manuály, elektronickou nápovědu v programu a případné další doprovodné texty.

## SHRNUTÍ

Tato kapitola je věnována popisu metodiky tvorby informačních systémů s využitím následující literatury:

## LITERATURA

**[Lef2010]** LEFFINGWELL, Dean. Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise**.** Addison-Wesley Professional, 2010,

**[Rat2011]** *Rational Unified Process: Best Practices for Software Development Teams***.** Rational Software. [cit. 2.4.2011]. Dostupný na WWW

**[ALD2005] ALDORF F.,** Metodika RUP - diplomová práce.

# 11 PRAKTICKÉ PŘÍKLADY VYUŽITÍ UML

V této kapitole jsou uvedeny příklady seminárních prací, které vznikly při výuce předmětu „Úvod do objektového modelování“. Dále jsou zde uvedeny příklady z různých zdrojů jako například ze školení produktu Enterprise Architect, Internetu apod. Kapitola poslouží studentům inovovaného předmětu „Metody objektového modelování“ jako inspirace při studiu a tvorbě seminárních prací.

## SEMINÁRNÍ PRÁCE

## Objednávka na e-shopu

Vypracoval: Lukáš Slechan

**Zadání - Téma**

V seminární práci bude popsán proces nakupování zboží na e-shopu prostřednictvím hned několika UML diagramů, mezi které patří diagram tříd, případu užití, aktivit a sekvenční diagram a jednom scénáři. Při tvorbě diagramů bude využito softwaru od společnosti Microsoft a to MS Visio 2010. Téma bylo zvoleno na základě toho, že nákup přes internet se stal běžnou záležitostí, a proto je vhodné se zamyslet na aktivitách, které zde probíhají.

Neregistrovaný zákazník (návštěvník)

Na stránky má volný vstup a není omezen z pohledu tvorby objednávky, ale pouze její modifikací. Tento typ uživatele může zboží vyhledávat, prohlížet, popřípadě přidávat do košíku, vyplnit a následně odeslat objednávku na zboží.

Registrovaný zákazník

Na rozdíl od návštěvníka má možnost spravovat svůj účet, kde si může prohlížet svou historii objednávek, modifikovat objednávku před její expedicí a hlavně možnost uplatnit na některé zboží slevový VIP program po překročení hranice souhrne útraty nad 2000Kč.

E-shop (systém)

Slouží k autentizaci a autorizaci zákazníků a evidenci a správě účtů včetně jejich objednávek. Disponuje příjemným uživatelským prostředím, které napomáhá k jednoduššímu nákupu zboží.

**Diagram tříd**

Diagram tříd zachycuje proces nákupu v e-shopu a znázorňuje třídy, které jsou využity v rámci systému objednávky. Daný diagram tříd obsahuje celkem šest tříd. Konkrétně se jedná o abstraktní třídu „Zákazník“, která je rodičem tříd „Neregistrovaný zákazník“ a „Registrovaný zákazník“. Ostatní třídy jsou „Autentizace“, „Objednávka“ a „Zboží“. V diagramu jsou znázorněny relace dědičnost, 1:1 a 1:*

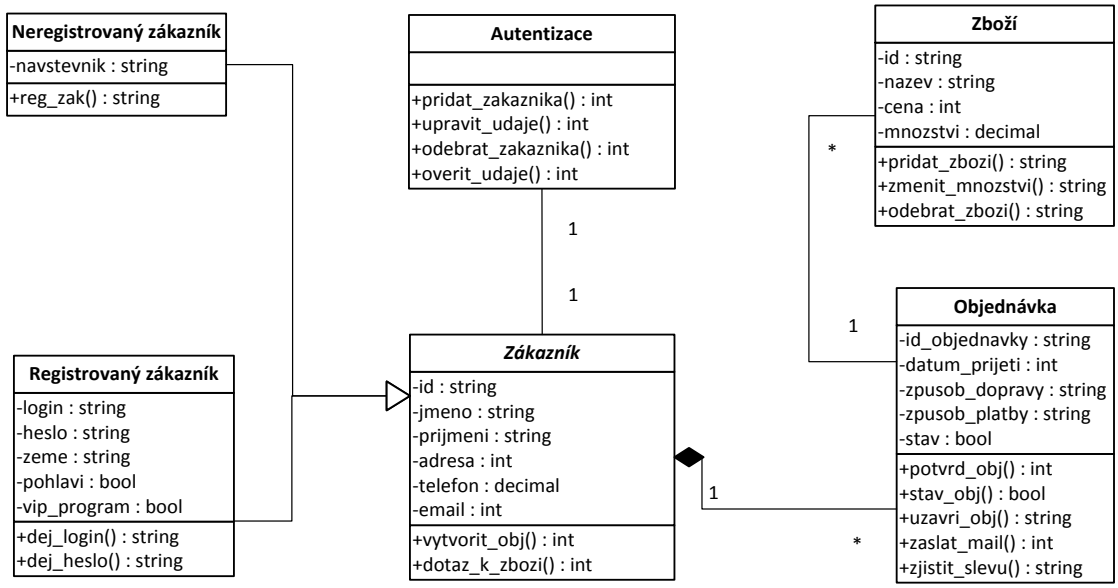
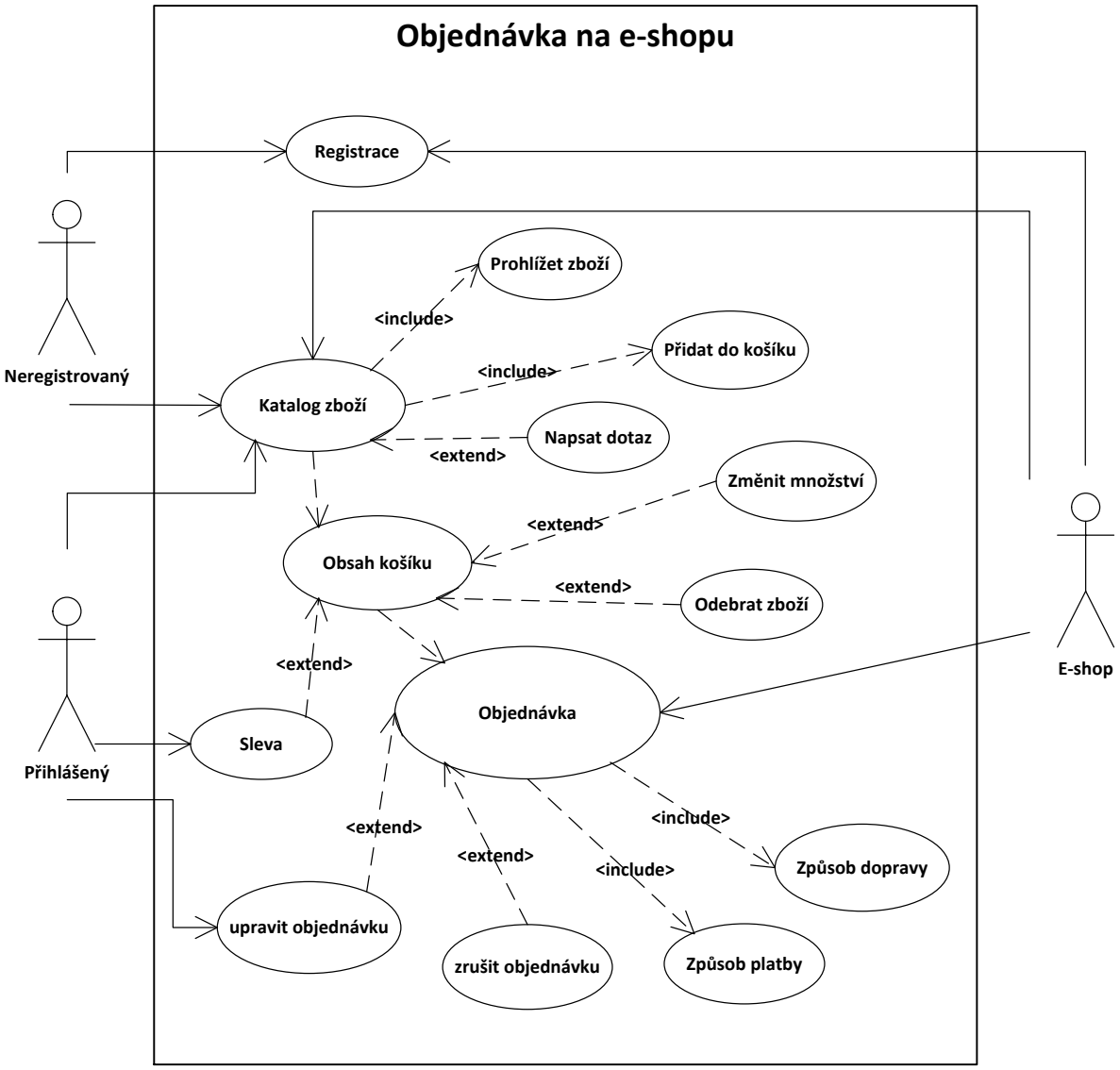



**Diagram případu užití**

V diagramu případu užití jsou zobrazeny 3 role: Neregistrovaný, Přihlášený a e-shop.
Neregistrovaný: Nemá povinnost se registrovat do systému a může objednávat zboží. Jeho nevýhodou je nemožnost uplatnění slevového VIP programu a musí vyplnit kontaktní údaje.
Přihlášený: Objednává zboží, na které může uplatnit slevový VIP program, sledovat svou historii objednávek nebo objednávku upravit.
E-shop (systém): Slouží pro registraci, přihlášení zákazníka. Provází jej napříč objednávkou.


Objednávka na e-shopu
Registrace
Prohlížet zboží
<include>
Přidat do košíku
<include>
Neregistrovaný
Katalog zboží
Napsat dotaz
<extend>
Změnit množství
<extend>
Obsah košíku
<extend>
Odebrat zboží
E-shop
<extend>
Objednávka
Sleva
Přihlášený
<extend>
<include>
<extend>
<include>
Způsob dopravy
upravit objednávku
zrušit objednávku
Způsob platby

## Diagram aktivit

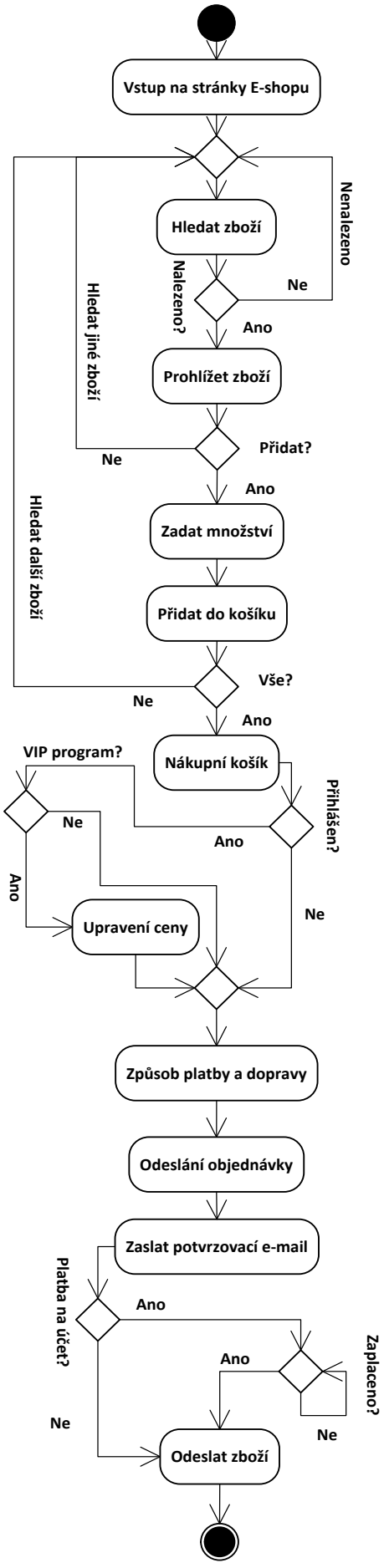
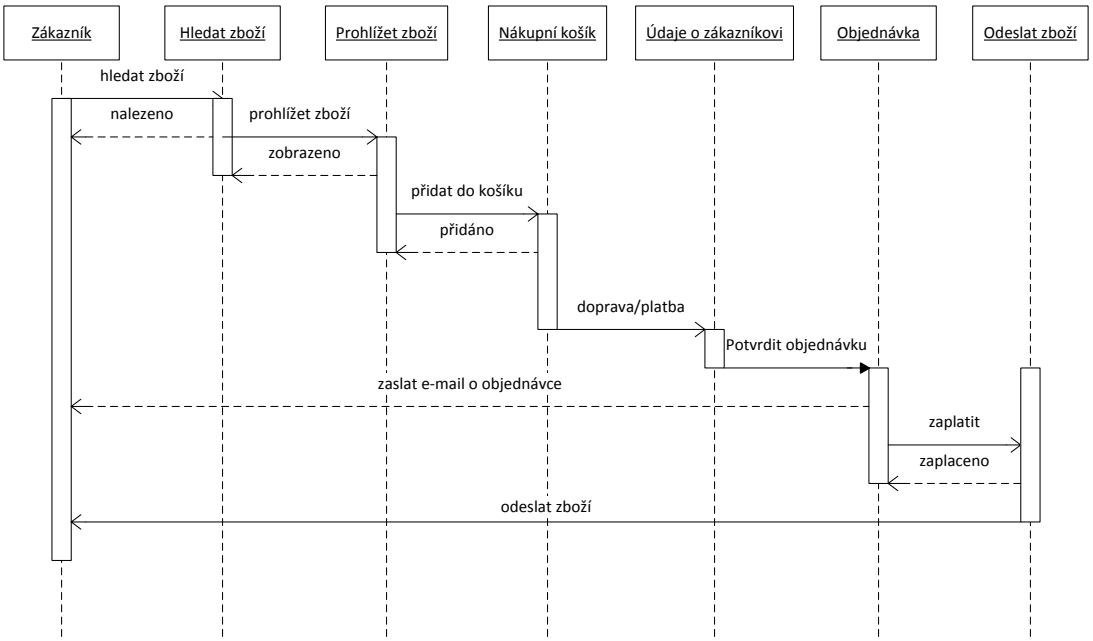


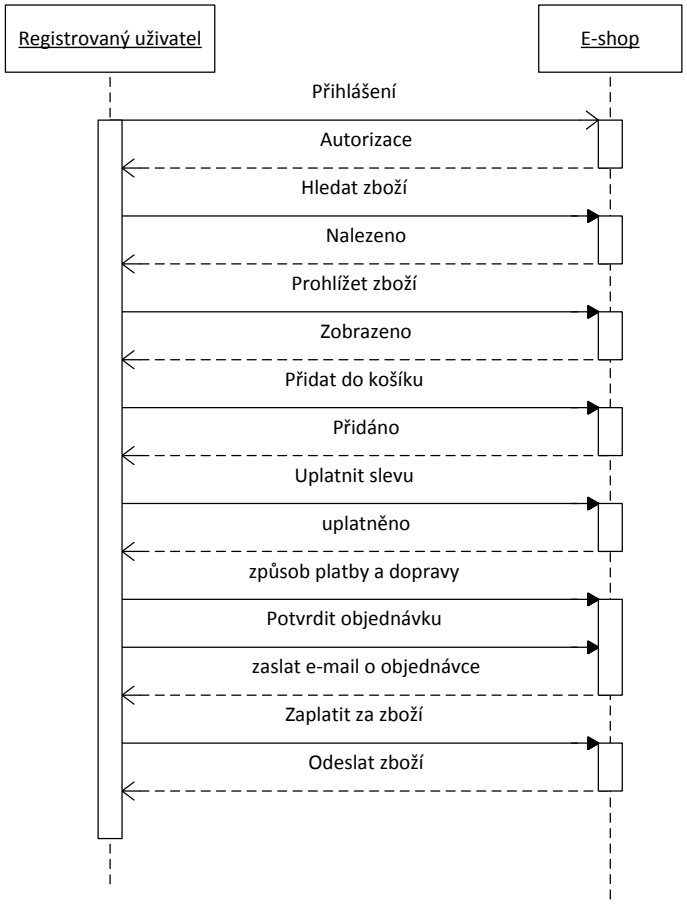

Diagram aktivit se používá pro modelování procedurální logiky procesů. Vytvořený diagram znázorňuje chování systému e-shopu při tvorbě objednávky zákazníkem. Diagram reprezentuje chování od vstupu na stránky, přes hledání zboží, prohlížení a mechanizmů při jeho koupi, tedy objednávky.

Proces začíná vstupem zákazníka na stránky e-shopu, kde začne hledat zboží dle parametrů. Jestliže systém toto zboží obsahuje, tak si jej zákazník prohlíží a případně přidá do nákupního košíku, kde podle potřeb může přidat i další zboží. Systém si před přechodem k dokončení objednávky zkontroluje, zda zákazník je přihlášen a má aktivovaný slevový VIP program. Pokud ano, tak systém upraví cenu. Následně je vyzván k vyplnění (v případě neregistrovaného zákazníka) nebo potvrzení (přihlášený zákazník) platebních, dopravních a doručovacích údajů. Následně je objednávka odeslána a systém zašle potvrzovací e-mail a na základě zvolené metodě platby odešle zboží.

**Sekvenční diagram - neregistrovaný uživatel**




**Sekvenční diagram - registrovaný uživatel**



### Scénář případu užití

Tato část se věnuje chronologickému popisu jednotlivým krokům při objednávání jediného zboží na e-shopu a to ve dvou případech. První případ se věnuje registrovanému uživateli a druhý zákazníkovi bez registrace.

**Tabulka 1 Případ registrovaného zákazníka, který uplatňuje slevu a platí dobírku**

| Krok | Role | Akce |
|---|---|---|
| 1 | Zákazník | Vstoupí na stránky e-shopu. |
| 2 | Zákazník | Přihlásí se do systému. |
| 3 | Systém | Provede autorizaci zákazníka. |
| 4 | Zákazník | Hledá v e-shopu zboží. |
| 5 | Systém | Pokud nalezne tak zobrazí zboží. |
| 6 | Zákazník | Prohlíží zboží. |
| 7 | Zákazník | Vybere si zboží a zadá množství. |
| 8 | Systém | Přidá zboží do košíku a zeptá se, zda chce pokračovat v nákupu nebo zaplatit. |
| 9 | Zákazník | Zákazník se rozhodne zaplatit. |
| 10 | Systém | Zkontroluje, zda zákazník má aktivován VIP program a případně upraví ceny. |
| 11 | Systém | Požádá zákazníka o kontrolu nákupního košíku. |
| 12 | Zákazník | Potvrdí obsah nákupního košíku. |
| 13 | Systém | Přesměruje zákazníka na způsob platby a dopravy. |
| 14 | Zákazník | Vybere si z příslušných možností metodu platby dobírkou a způsob dopravy. |
| 15 | Systém | Zaeviduje a požádá zákazníka o potvrzení popřípadě o změnu dodacích údajů. |
| 16 | Zákazník | Potvrdí dodací údaje a odešle objednávku. |
| 17 | Systém | Zaeviduje objednávku a zašle potvrzovací e-mail. |
| 18 | Systém | Odešle zboží zákazníkovi. |

**Tabulka 2 Případ neregistrovaného zákazníka, který zvolil platbu přes bankovní účet**

| Krok | Role | Akce |
|---|---|---|
| 1 | Zákazník | Vstoupí na stránky e-shopu. |
| 2 | Zákazník | Hledá v e-shopu zboží. |
| 3 | Systém | Pokud nalezne tak zobrazí zboží. |
| 4 | Zákazník | Prohlíží zboží. |
| 5 | Zákazník | Vybere si zboží a zadá množství. |
| 6 | Systém | Přidá zboží do košíku a zeptá se, zda chce pokračovat v nákupu nebo zaplatit. |
| 7 | Zákazník | Zákazník se rozhodne zaplatit. |
| 8 | Systém | Požádá zákazníka o kontrolu nákupního košíku. |
| 9 | Zákazník | Potvrdí obsah nákupního košíku. |
| 10 | Systém | Přesměruje zákazníka na způsob platby a dopravy. |
| 11 | Zákazník | Vybere si z příslušných možností metodu platby z účtu a způsob dopravy. |
| 12 | Systém | Zaeviduje a požádá zákazníka o vyplnění dodacích údajů. |
| 13 | Zákazník | Vyplní dodací údaje a odešle objednávku. |
| 14 | Systém | Zaeviduje objednávku a zašle potvrzovací e-mail s informacemi o účtu. |
| 15 | Zákazník | Zaplatí za objednávku. |
| 16 | Systém | Zaeviduje platbu a odešle zboží zákazníkovi. |

# SEMINÁRNÍ PRÁCE

## Téma: Import zboží do IS společnosti eShopSystem

Vypracoval: Lukáš Dziadek

**Import zboží do IS společnosti eShopSystem**

Společnost eShopSystem se věnuje tvorbě internetových prezentací a obchodů. Obsah webových stránek je spravován pomocí administračního rozhraní. Pro usnadnění práce systém poskytuje možnost automatických importů. Slovo import je v této souvislosti možno chápat jako posloupnost činností, která vede k tomu, že se v systému aktualizuje stávající zboží, vloží nové a odstraní již neaktuální zboží.

Tato seminární práce se pokusí ve zjednodušené podobě popsat obecný princip importu položek do systému. Na jednotlivých diagramech bude názorně ukázáno, jak by měl programátor implementovat svůj program, který bude zpracovávat data od určitého dodavatele.

V praxi je spuštěno již několik desítek takových skriptů, které den co den provádějí aktualizaci dat v IS. V dnešní době je prakticky nemožné, aby tuto činnost vykonával člověk ručně. Když si představíme modelovou situaci na internetovém obchodu s pneumatikami, tak zjistíme, že počet položek středně velkého obchodu je 10 - 30 tisíc. Není tedy možné provádět aktualizaci skladových zásob, cen a dalších údajů ručně – respektive bylo by to finančně neúnosné.

**Diagram tříd**

Diagram tříd bude znázorňovat třídy, které se v importu využívají. Jmenovitě se jedná o třídu Zboží, která se stará o vytváření, editaci a mazání jednotlivých záznamů z DB. Dále o třídu Import, která řeší samotné zpracování dat. A v neposlední řadě o třídu Systém, která zajišťuje různé pomocné akce jako je odeslání emailu nebo logování výstupu.

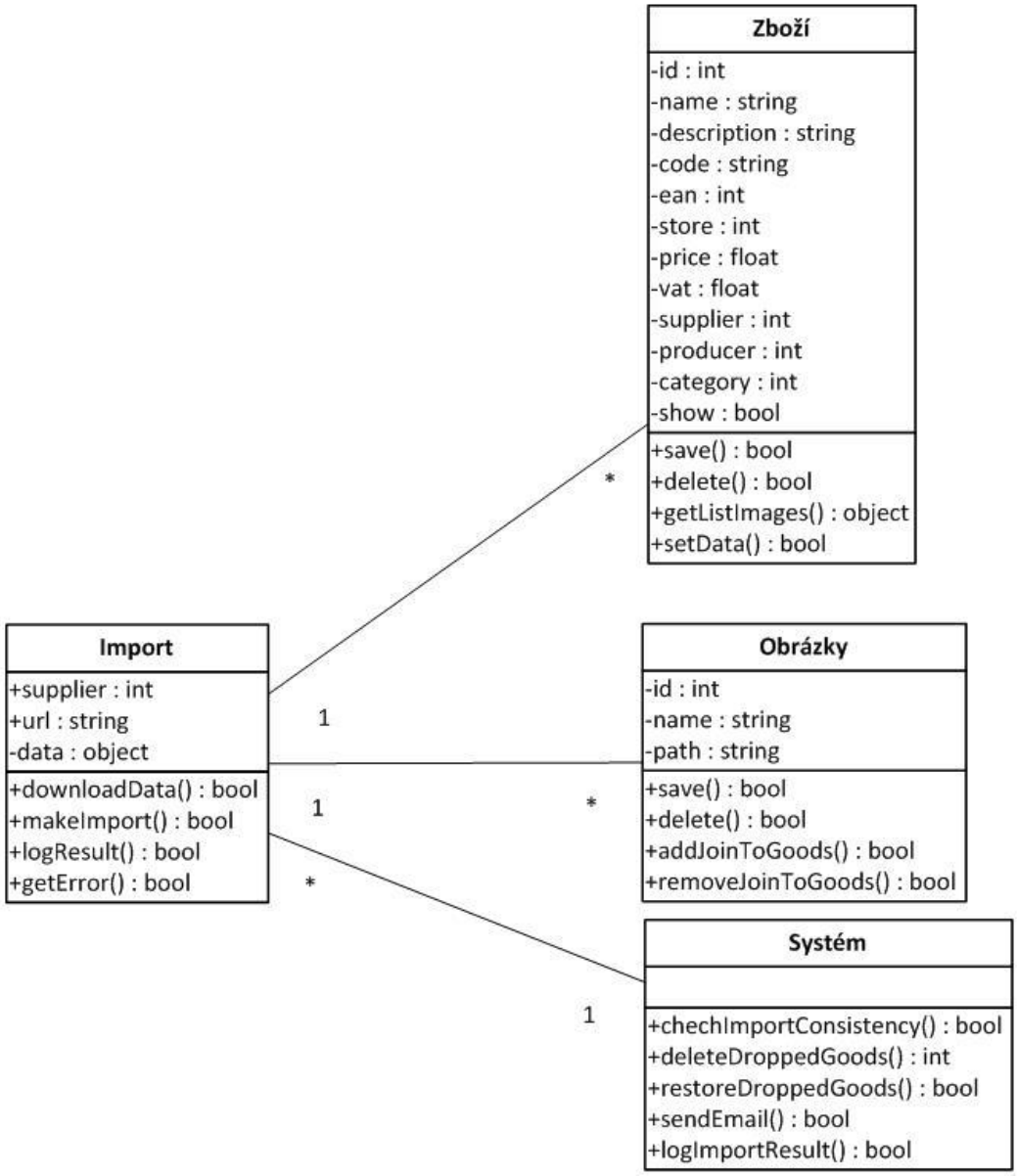
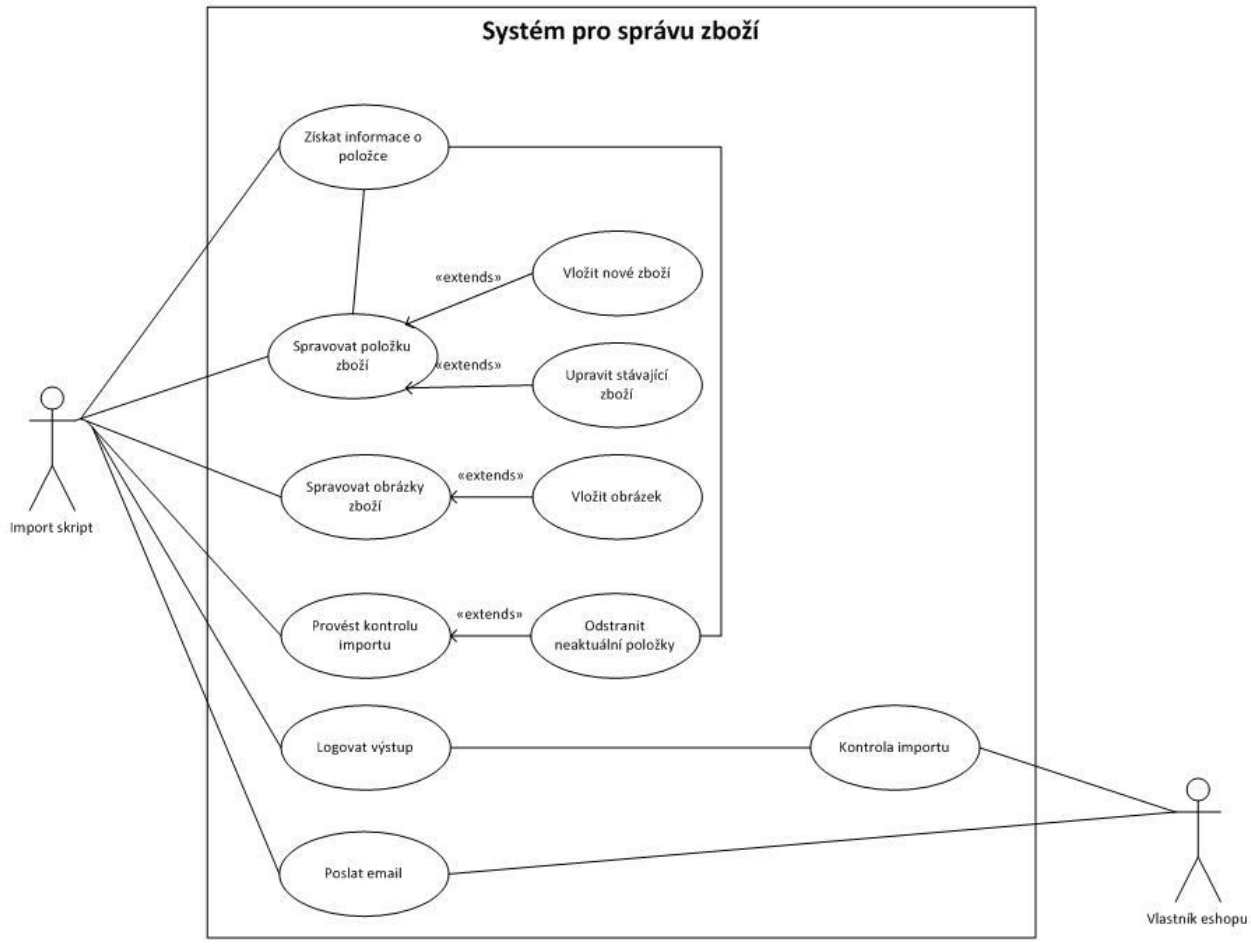


**Diagram případu užití**

V diagramu jsou zobrazeny 2 role: import skript a administrátor.

**Import skript** – jedná se o roli, která prezentuje samotný skript, jenž vkládá zboží automatizovaně do systému. Možná akce jsou: získání informací o zboží, správa položky zboží (vytvoření a editace), správa obrázku k danému zboží, provádění kontroly importu, logování výstupu importu a poslání informačního emailu administrátorovi.

**Administrátor** – pro potřeby v tomto případě má dostupnou jedinou akci: kontrolu importu.



**Sekvenční diagram**

Diagram popisující sekvenci činností – to jak mají jít jednotlivé činnosti za sebou, aby byl dodržen logický koncept importu. Vynechání některého bodu může mít za následek nesprávné chování systému.

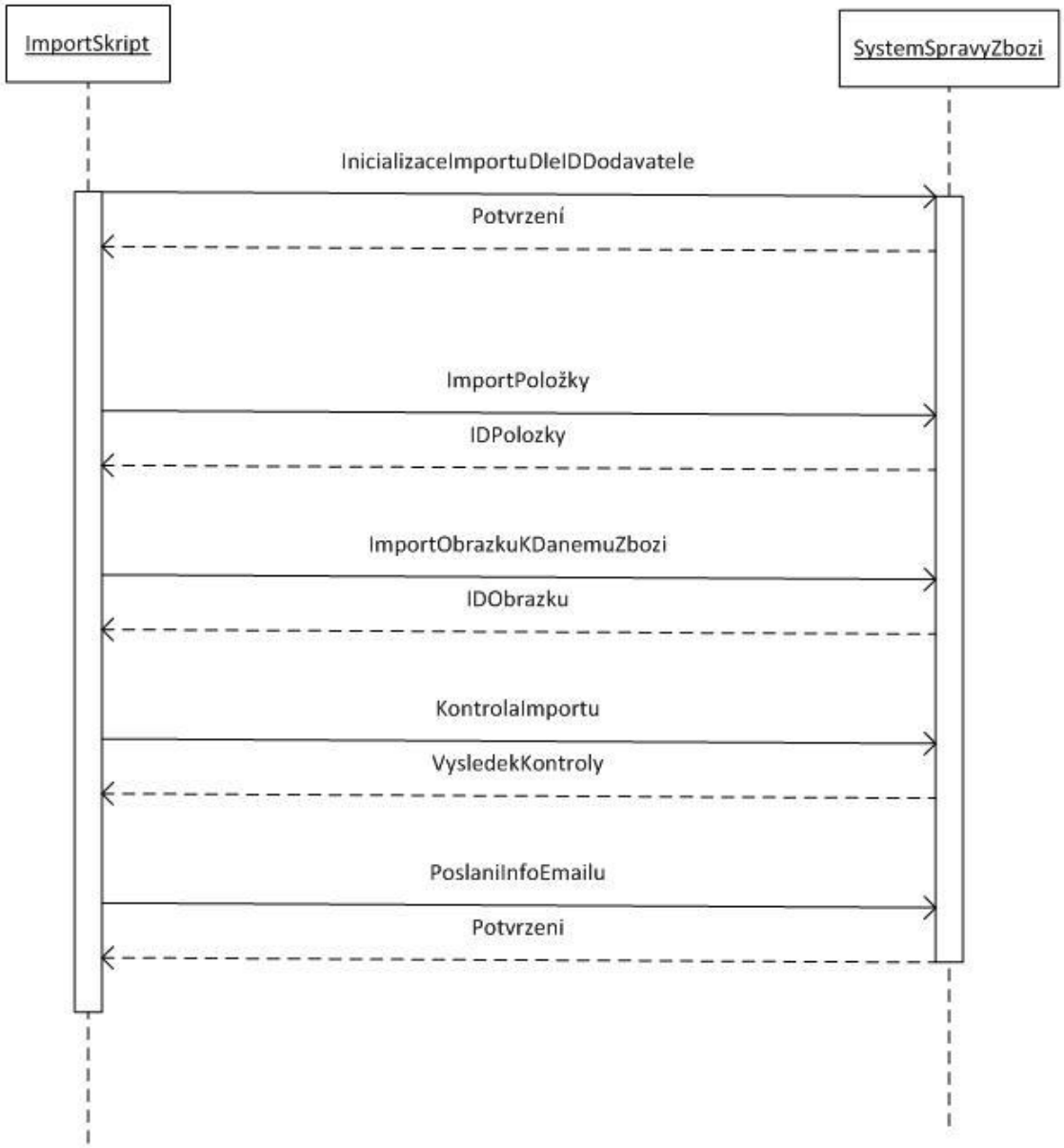
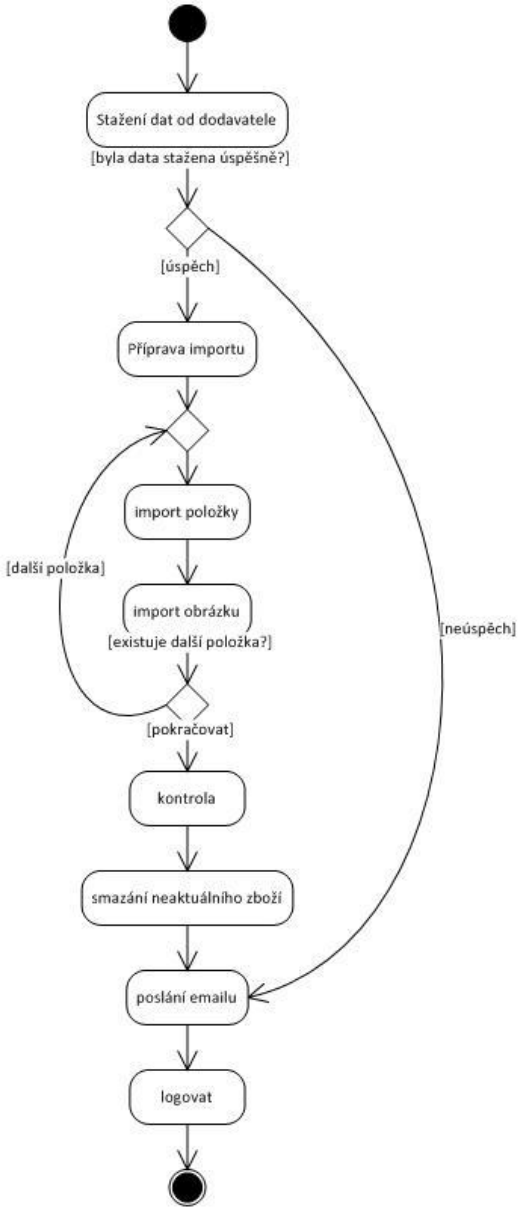


**Diagram aktivit**

Diagram popisu logickou vrstvu aplikace. Je zde názorně vidět hlavní body, ve kterých se import skript bude rozhodovat, co má dále dělat. Jde zejména o ošetření chybového stavu, když se nepodaří stáhnout zboží a potom o smyčku, ve které se importuje zboží.



# SEMINÁRNÍ PRÁCE

## Registrace do eshopu a nákup motoru

**Vypracoval: Michal Janoušek**

**Téma**

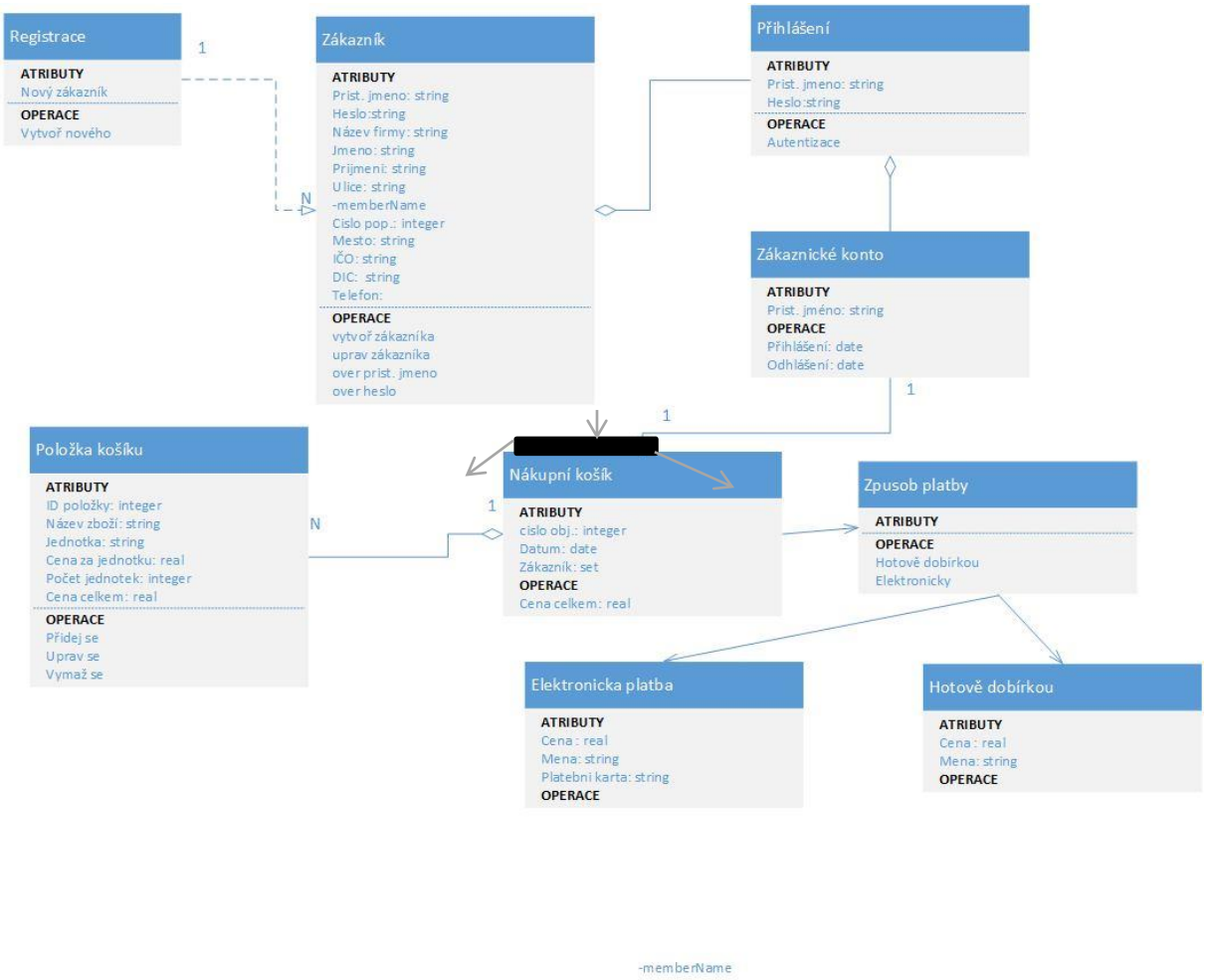
Tato seminární práce modeluje posloupnost operací při registraci do eshopu zdz.cz a nákupu motoru v tomto eshopu. Graficky bude tato práce vycházet ze tří diagramů a sice diagramu tříd, diagramu případu užití a diagramu aktivit. Toto téma bylo zvoleno z důvodu znalosti daného eshopu a systému jeho fungování. Model vychází z jednotlivých operací při registraci, jejím autorizování a následném nákupu motoru.

Diagramy jsem tvořil v programu Visio professional.

## Diagram tříd

Na níže uvedeném diagramu tříd jsou zachyceny jednotlivé třídy objektů, jejich atributy a operace. Znázorňuje vazby mezi třídami případně dědičnost. Struktura třídy jež je definována jako její atributy , operace a případně omezení, tvoří předpis pro tvorbu reálných objektů v budoucím fungujícím informačním systému.

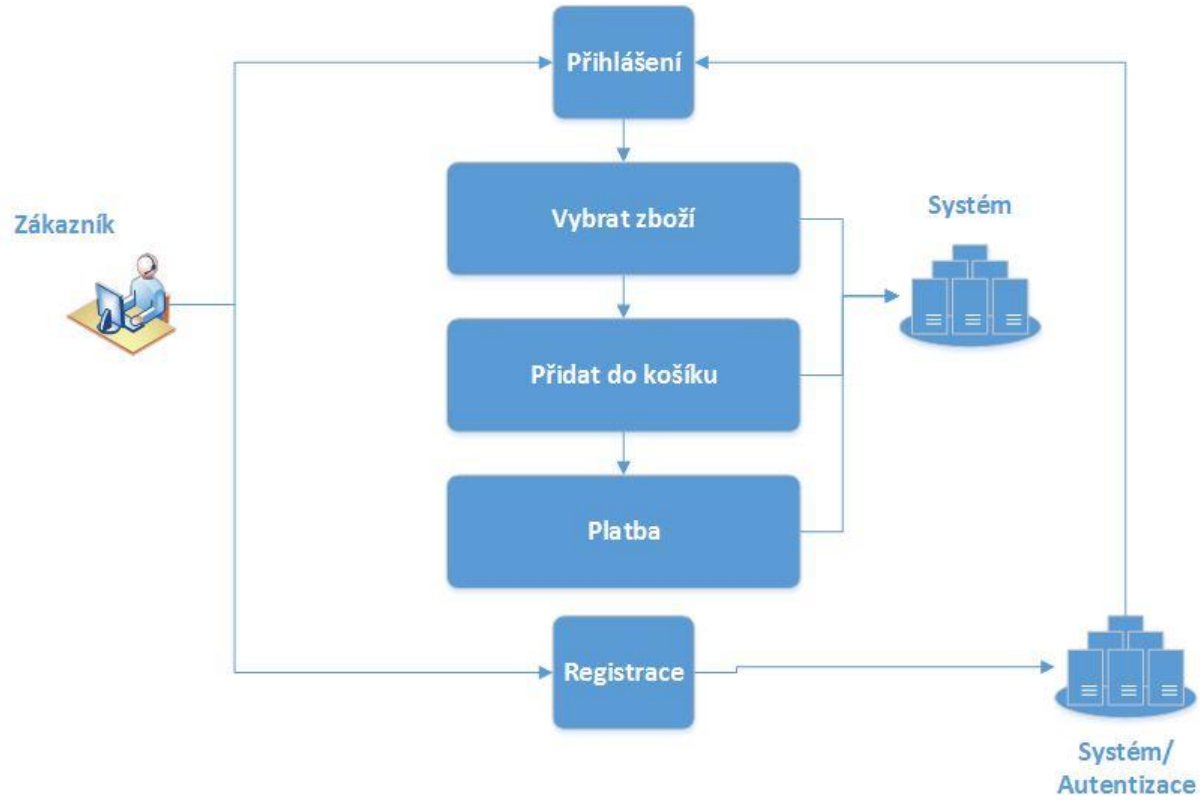
**Obrázek 36: Diagram tříd**



### Diagram případu užití

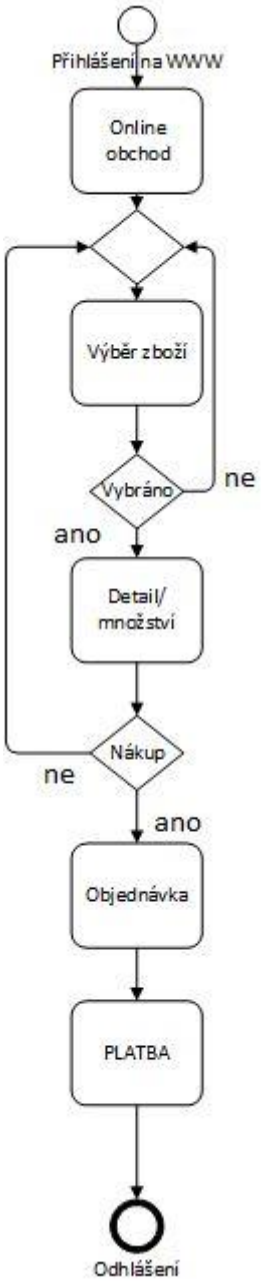
Diagram případu užití zachycuje detailně funkční chování budoucího informačního systému a jednoznačně vymezuje funkčnost po něm požadovanou zadavatelem. Jelikož tento diagram přesně vymezuje rozsah budoucích vlastností a podle něho se budou tyto programovat. Je nutné Detailně promyslet a navrhnout v tomto diagramu požadované chování systému za všech přepdpokládatelných okolností. Systém nebude umět to, co v diagramu případů užití není definováno. Aktér v našem případě zákazník vystupuje vůči systému ze své role a má předem vymezenu množinu případů užití. Tyto případy užití jsou společné více aktérům, zákazníkům. Systém vystupuje také jako aktér, má však nadefinovány jiné případy užití.

**Obrázek 37: Diagram případů užití**




### Diagram aktivit

Diagram aktivit graficky zobrazuje jednotlivé události na straně aktéra či systému a lze jimi zachytit a popsat jak prosté sekvenční, tak paralelní chování s podmíněným větvením. Elementy diagramů aktivit použité na obrázku níže jsou tyto: akce, přechod, hodnocení přechodů, větvení a spojení.




**Závěr**

V rámci této práce jsme na praktickém případě vyzkoušeli a realizovali popis jednoduchého systému. Samostatná aplikace znalostí nabytých v rámci přednášek a cvičení, a to jak při vlastní logice tvorby diagramů, tak použití MS Visio nebyla jednoduchá, ale pro pochopení daného učiva zcela nezbytná. Praktické použití základů objektově orientovaného programovacího jazyka UML je zcela nezbytné při komunikaci mezi zadavatelem software, zákazníkem a programátory a pomáhá vizualizovat vlastnosti systému tak, aby byly pochopitelné a srozumitelné pro všechny na projektu zúčastněné a nedocházelo k nejednoznačnostem v chápání funkčních vlastností navrhovaného systému.

# SEMINÁRNÍ PRÁCE

## Poštové služby

Vypracoval: Pavol Cvinček

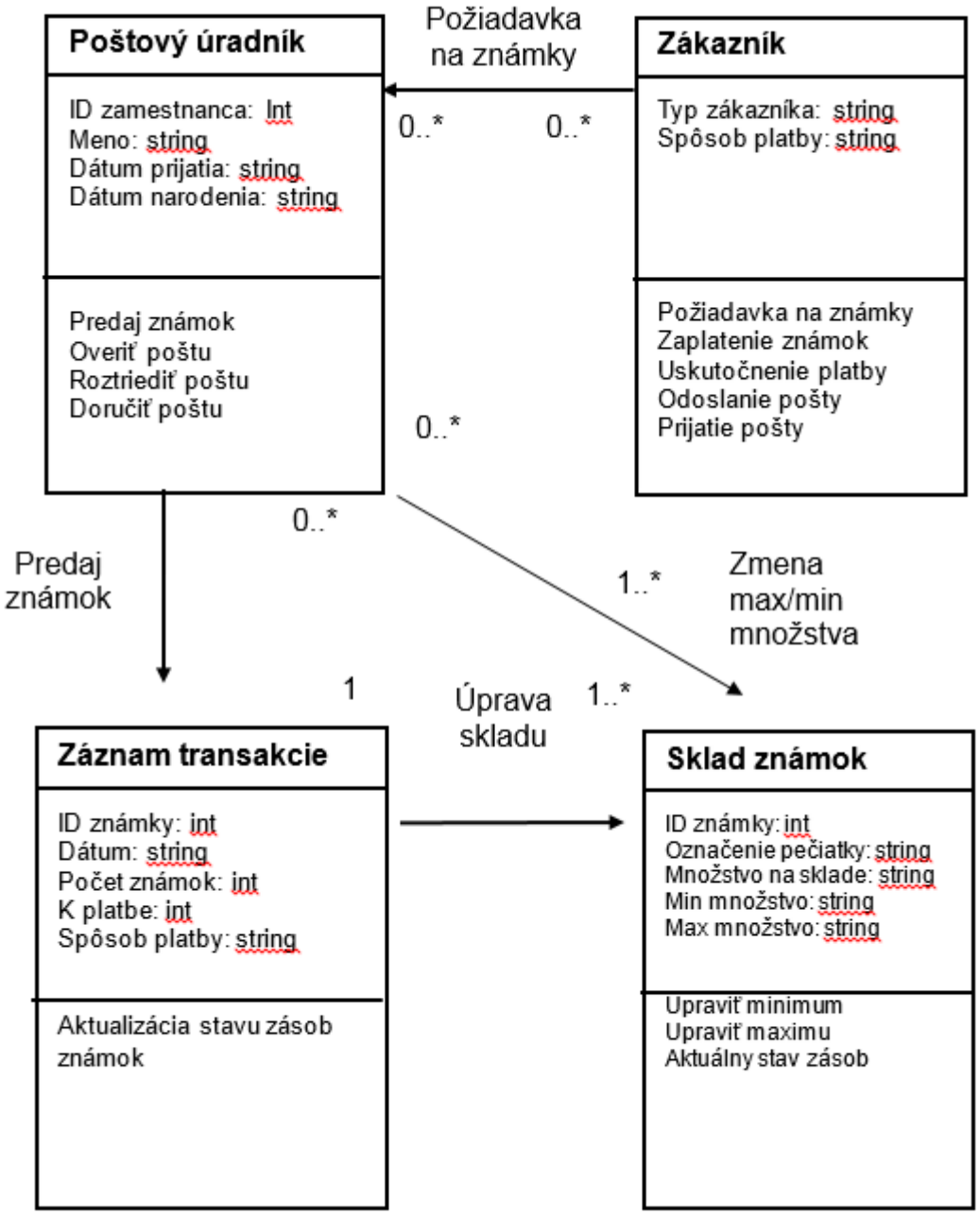
**Zadání - Téma**

Tento informačný systém má popisovať zjednodušené fungovanie pošty. Jedná sa predovšetkým o posielanie resp. prijímanie listových zásielok. S týmto bezpodmienečne súvisí predaj známok, ktoré sú nevyhnutné pre posielanie pošty. Bližší popis poskytuje priložená tabuľka.

| | |
|---|---|
| Meno: | Nákup známok, Odoslanie/Prijatie pošty |
| Aktéri: | Zákazník, Poštový úradník |
| Popis: | Tento prípad užitia popisuje proces nákupu známok a prijatie/odoslanie pošty |
| Úspešné ukončenie: | 1. Zákazník žiada známky<br>2. Poštový úradník skontroluje dostupnosť známok na sklade<br>3. Známky sú dostupné a zákazník zaplatí<br>4. Zákazník prijíma známky a následne je upravený stav zásob<br>5. Zákazník odosiela poštu |
| Alternatívne riešenie: | 1. Zákazník žiada známky<br>2. Poštový úradník skontroluje dostupnosť známok na sklade<br>3. Známky nie sú dostupné<br>4. Zákazník odchádza bez známok<br>5. Zákazník nezaplatil za služby<br>6. Neprebieha odoslanie pošty |
| Počiatočný stav: | Zákazník chce kúpiť známky a odoslať poštu |

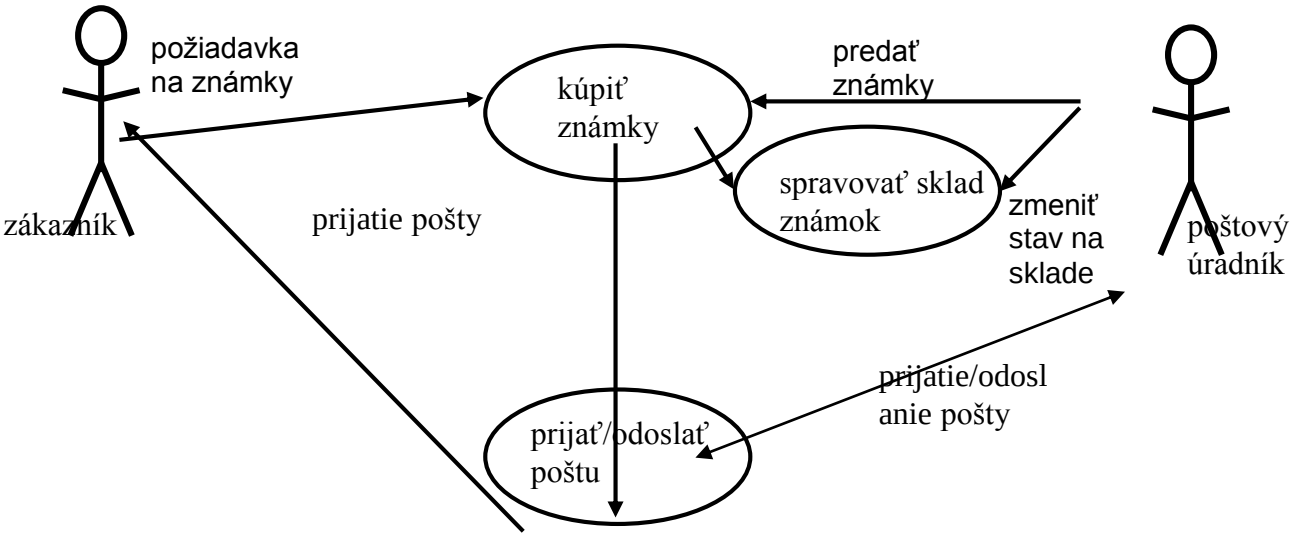
**Diagram tried**

Tento informačný systém využíva 4 základné triedy, a to triedy Poštový úradník, Zákazník, záznam transakcie a Sklad známok. Tieto triedy majú svoje atribúty a metódy, všetko je popísané v nasledujúcom diagrame.



**USE case diagram pre poštové služby**

Nasledujúci diagram užitia zobrazuje celý proces odoslania pošty od zákazníka cez poštového úradníka až k samotnému odoslaniu poštovej zásielky.




**4 Sekvenčný diagram poštových služieb**

Nasledujúci diagram zachytáva danú problematiky po jednotlivých krokoch.

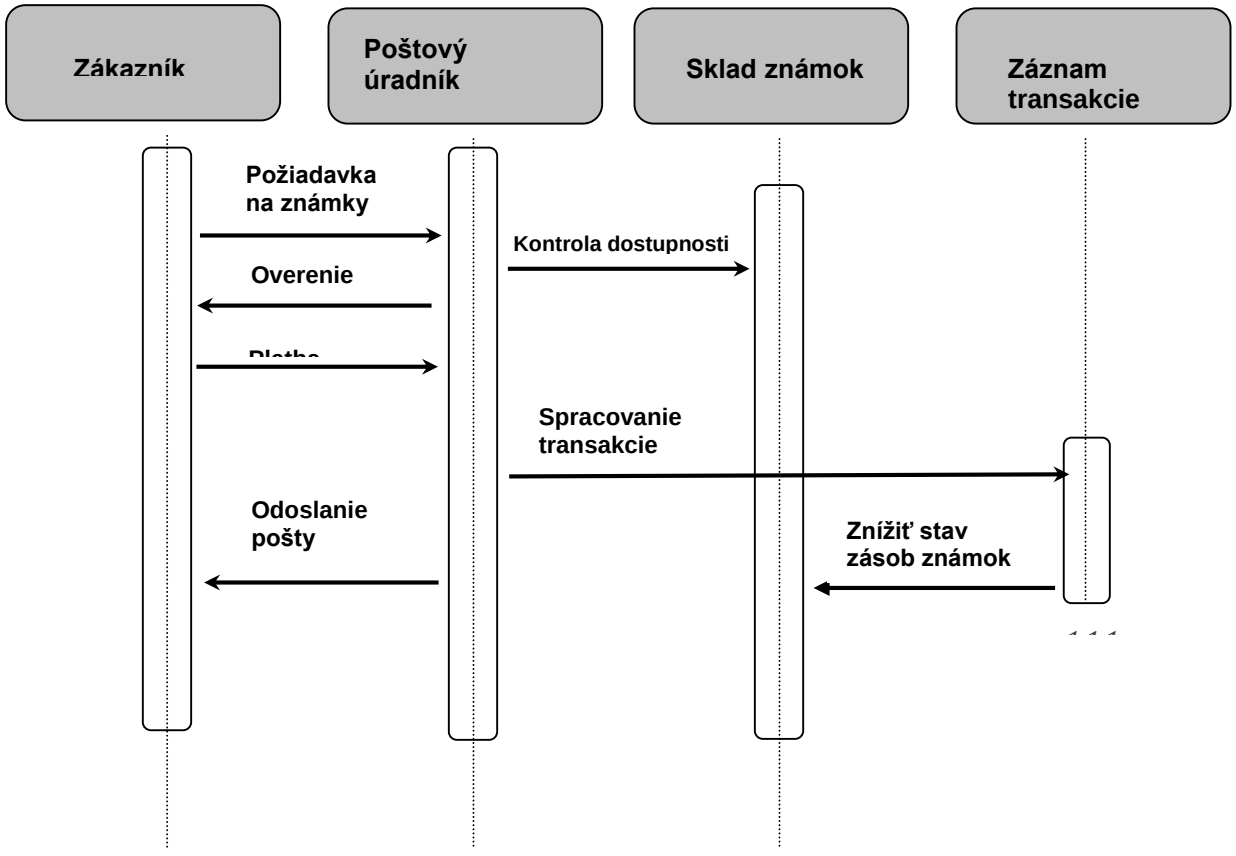
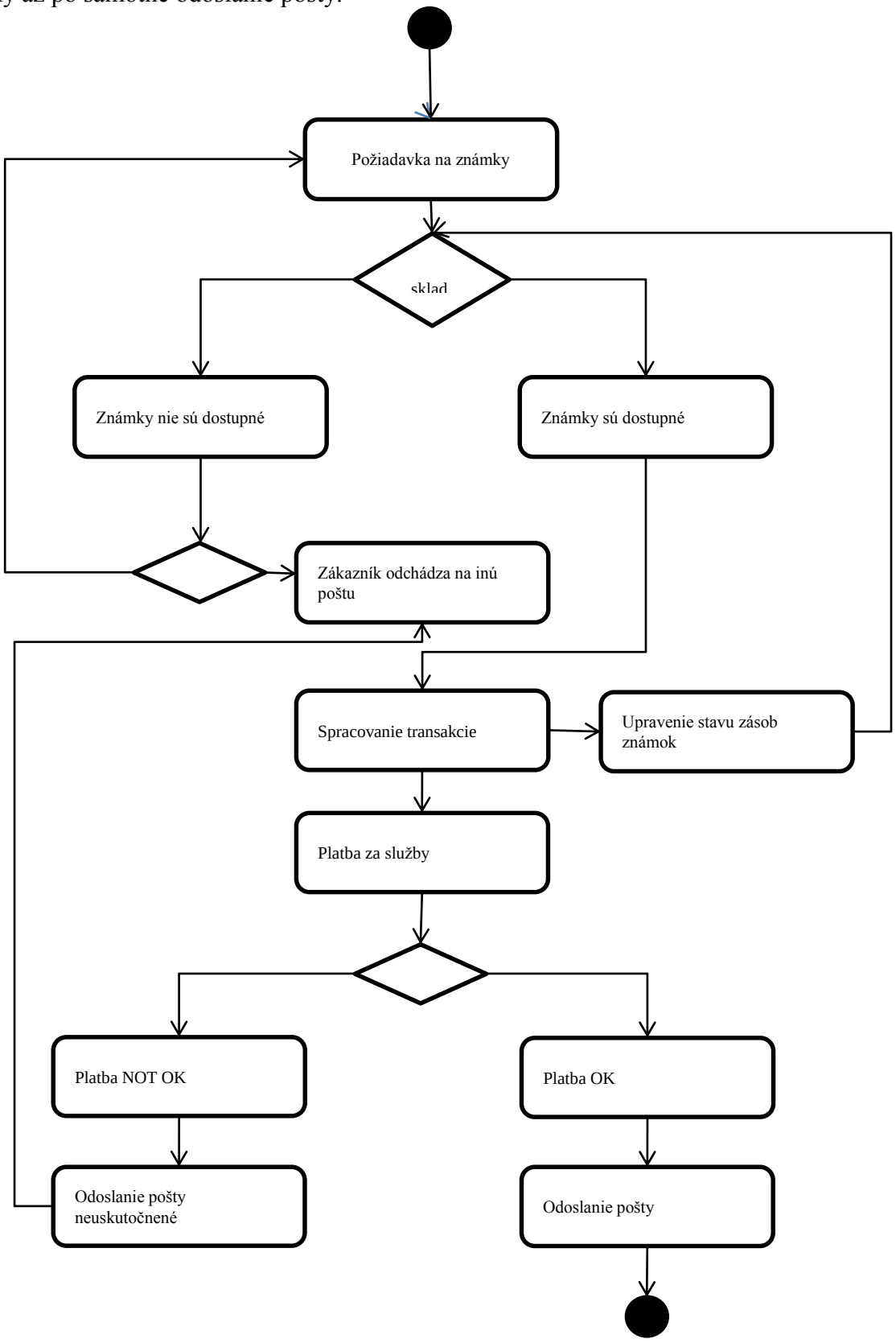


**Diagram aktivít**

Tento diagram zobrazuje celú cestu odoslania poštovej zásielky od nákupu poštovej známky až po samotné odoslanie pošty.



# SEMINÁRNÍ PRÁCE

## IS Svaz

**Vypracoval: Václav Stach**

### Zadání - Téma

Navrhovaný informační systém má sloužit k potřebám krajských fotbalových svazů, které se starají pouze o výkonnostní ( amatérský ) fotbal, jeho soutěže a hráče. Účelem zavedení systému je nahrazení starých kartotékových pořadačů a systémů založených na systému tužka-papír novými informačními technologiemi, jenž by nejen zrychlili a ulehčili práci personálu, ale zabránili by i některým chybám vznikajících například přehlédnutím redundance dat a dalších chyb vznikajících přirozeným lidským spravováním takto obsáhlých databází.

Přesto tento systém neumožní vyřízení celého procesu mezi kluby a svazem pouze online cestou, neboť registrace hráčů musí být zalita v pevném obalu a musí obsahovat razítko a další znaky zabraňující zfalšování. Stejně tak všechny ostatní listiny potřebné pro provedení procesů a registrací zatím neumožňují online ověření, proto i tato komunikace musí probíhat poštou.

Tento informační systém přinese klubům:

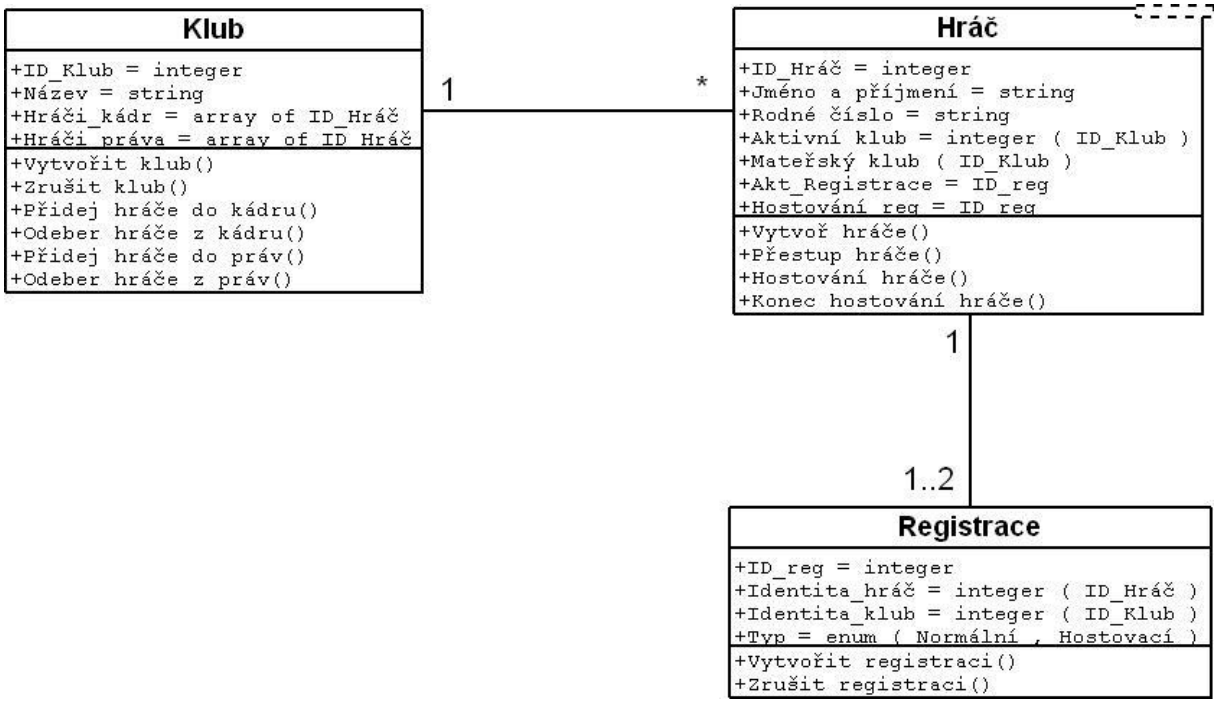
- Rychlé a především aktuální informace o registrování hráčů
- Možnost informovat se, v jakém stádiu je jejich žádost
- Možnost rychlého ověření vlastního kádru v případě nejasností

Fotbalovému svazu přinese IS tyto výhody:

- Zabránění redundance dat
- Efektivnější a rychlejší vyřízení jednotlivých úkonů
- Rychlejší vyhledávání všech provedených operací
- Možnost vyřešení drobných nesrozumitelností s klubem online ( dnes poštou )
- V budoucnu možnost vyřízení celé transakce internetovou cestou

**Diagram tříd**

Tento informační systém využívá tři základní třídy, a to třídy Klub, Hráč a Registrace. Tyto mají své atributy a metody, vše je popsáno v následujícím diagramu.




**USE CASE Diagram**

**Evidence žádostí**

Pracovníkovi je zobrazen formulář žádostí, do které uvede všechny údaje týkající se požadovaného úkonu, systém okamžitě ověří správnost údajů a pošle žádost do příslušné fronty na vyřízení úkonu ( Evidence hráčů, Evidence Klubů )

**Evidence hráčů**

Podle žádosti která přišla rozhodne personál o následující akci, na formuláři budou zobrazeny veškeré informace o daném hráči a jeho stavu z pohledu svazu, tak i možnost otevření informací ohledně týmů zapojených do transakce.

**Evidence klubů**

Pokud přijde žádost o zrušení či vytvoření klubu, bude zde zobrazen formulář pro vyplnění a zadání všech potřebných informací. Ostatní funkce jsou odvislé od jiných požadavků v Evidenci hráčů.

**Evidence registrací**

V této evidenci je možno prohlédnout si jakékoliv údaje o dané registraci a provést její zrušení, nebo naopak vytvoření nové registrace, obě tyto akce jsou však podmíněny jinými akcemi v Evidenci hráčů.

**Use Case Diagram**

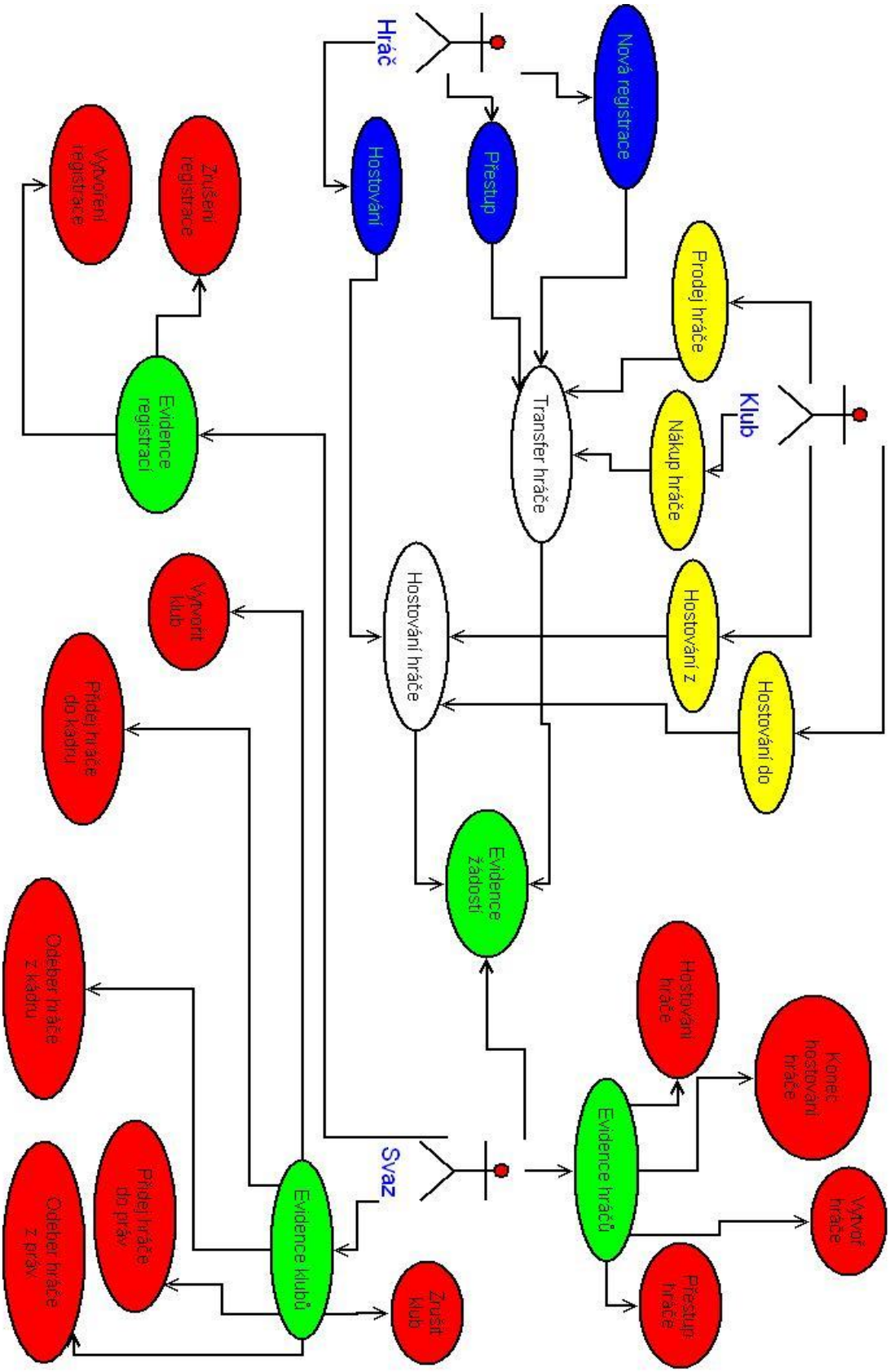
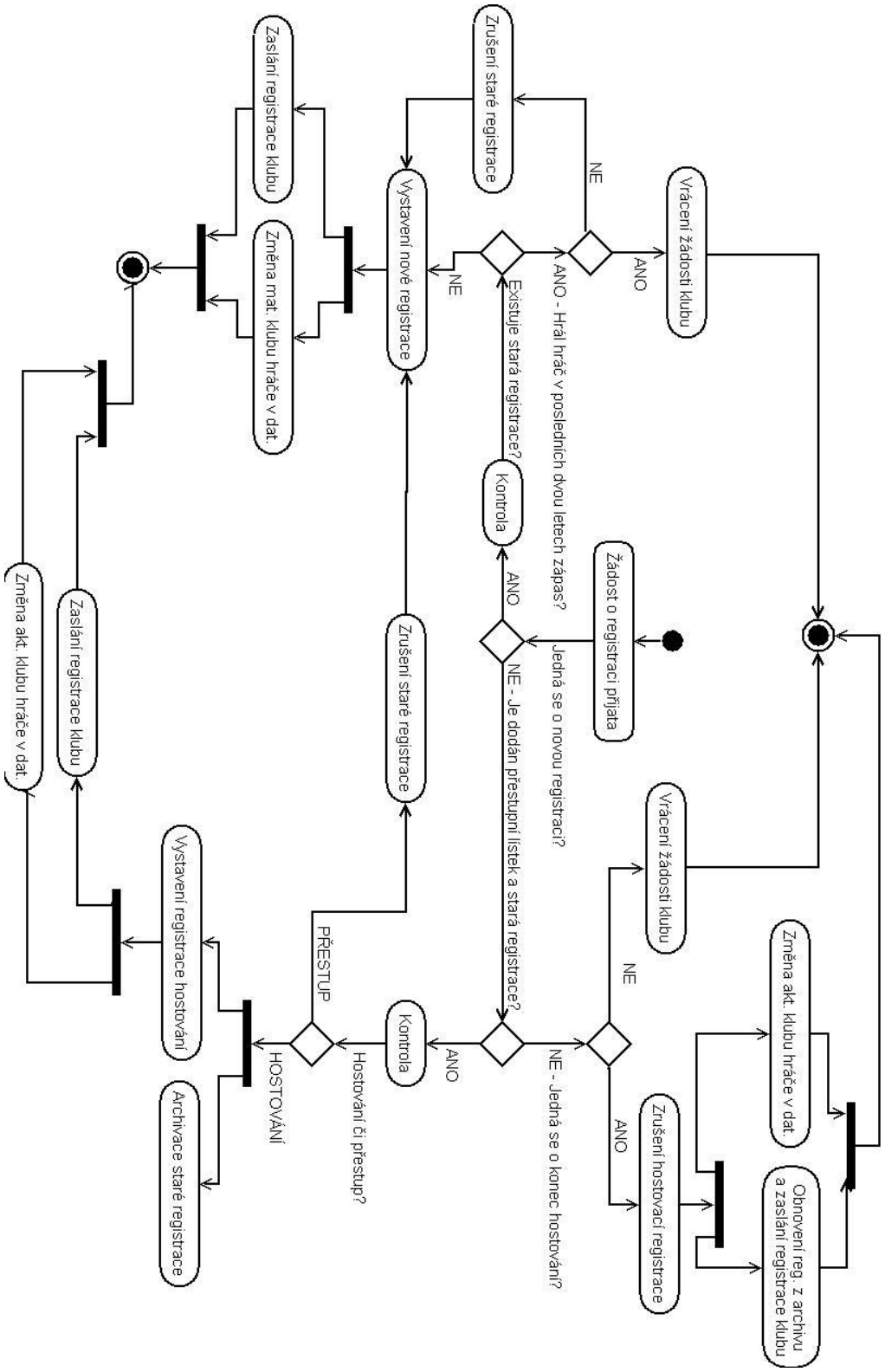

Hráč
Nová registrace
Hostování
Přestup
Vytvoření registrace
Zrušení registrace
Prodej hráče
Klub
Transfer hráče
Nákup hráče
Evidence registrací
Hostování hráče
Hostování z
Hostování do
Vytvořit klub
Přidej hráče do kádru
Evidence žádostí
Odeber hráče z kádru
Hostování hráče
Konec hostování hráče
Svaz
Evidence hráčů
Vytvoř hráče
Přestup hráče
Přidej hráče do práv
Evidence klubů
Odeber hráče z práv
Zrušit klub

**Diagram činností**

Tento diagram ukazuje celou cestu, kterou probíhá rozhodování od přijmutí žádosti o registraci či jiného úkonu, až po ukončení procesu ať už formou nevyřízení, či vyřízení požadavku.



**Scénář případu užití.**

Tento scénář ukazuje, jaká interakce probíhá mezi obsluhou a systémem v případě, že se snaží provést Hostování hráče.

| Krok | Role | Akce |
|---|---|---|
| 1 | Obsluha | Zadá pokyn k provedení hostování hráče |
| 2 | Systém | Zobrazí seznam hráčů |
| 3 | Obsluha | Vybere požadovaného hráče |
| 4 | Systém | Zobrazí všechny atributy daného hráče |
| 5 | Obsluha | Spustí funkci Hostování hráče |
| 6 | Systém | Uloží do atributu Hostování_reg ID aktuální registrace a zobrazí formulář hostování |
| 7 | Obsluha | Zadá klub, do kterého má jít hráč hostovat a potvrdí |
| 8 | Systém | Změní atribut Aktivní klub na ID vybraného klubu ; spustí fci Vytvoř registraci, jenž přiřadí registraci nové jedinečné číslo registrace, atributu ID_hráč přiřadí ID aktuálního hráče, do ID_klub přiřadí ID cílového klubu, do atributu Typ nastaví hodnotu Hostovací ; spustí fci Odeber hráče z kádru pro Mateřský klub a odebere ID_hráče ; spustí fci Přidej hráče do kádru a přidá do pole Hráči_kádr ID hráče na hostování |
| 9 | Obsluha | Stiskne Vytisknutí registrace |
| 10 | Systém | Odešle na frontu tiskárny formulář pro vytisknutí Hostovací registrace |
| 11 | Obsluha | Stiskne Dokončení operace |
| 12 | Systém | Zašle automatický e-mail na adresu klubu o dokončení operace s přibližným datem příchodu registrace přes poštu ; Navrátí obrazovku do původního menu |

# SEMINÁRNÍ PRÁCE

## Návrh aplikace pro optimalizaci procesu vyskladnění

**Vypracoval: Jan Dittrich**

**Téma - zadání**

Tato seminární práce, vznikla jako základní koncept pozdější práce diplomové. Text a diagramy níže uvedené, jsou návrhem aplikace pro optimalizaci vyskladňování ve společnosti Gates.

**Určení rolí**

S aplikací budou v první fázi pracovat 3 uživatelé. Jejich role jsou následující:


Administrátor

Má přístup ke všem částem systému. V jeho pravomoci je upravit interní parametry systému (např. číselníky), pracovat v systému (vytvářet nové nákladky a nákladní listy, popřípadě je editovat) a zobrazit databázi výstupů.

Vedoucí skladu

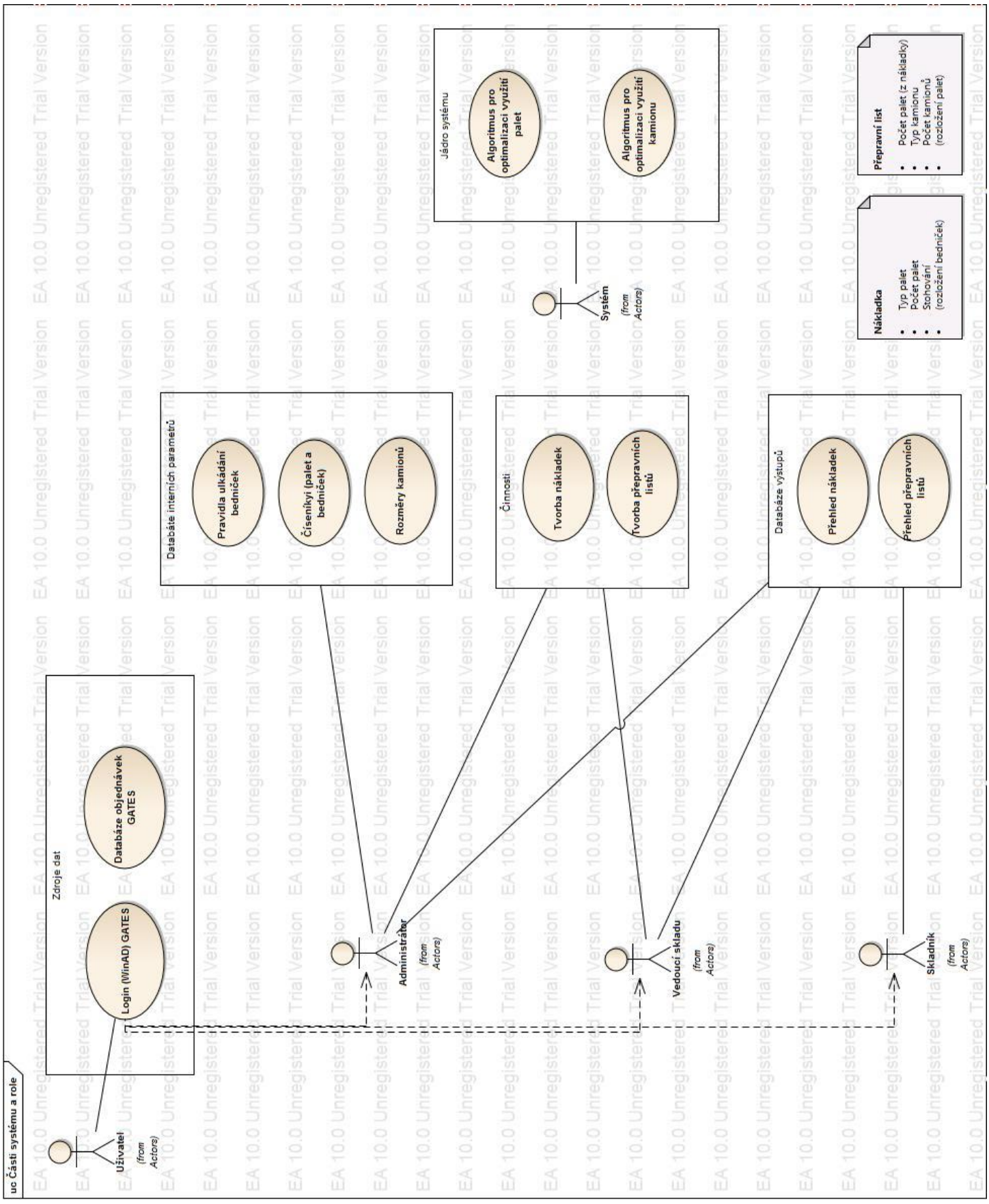
Osoba, pro kterou je systém primárně určen. Může vytvářet nové a editovat stávající nákladky a nákladní listy, rovněž může zobrazit databázi již vytvořených.

Skladník

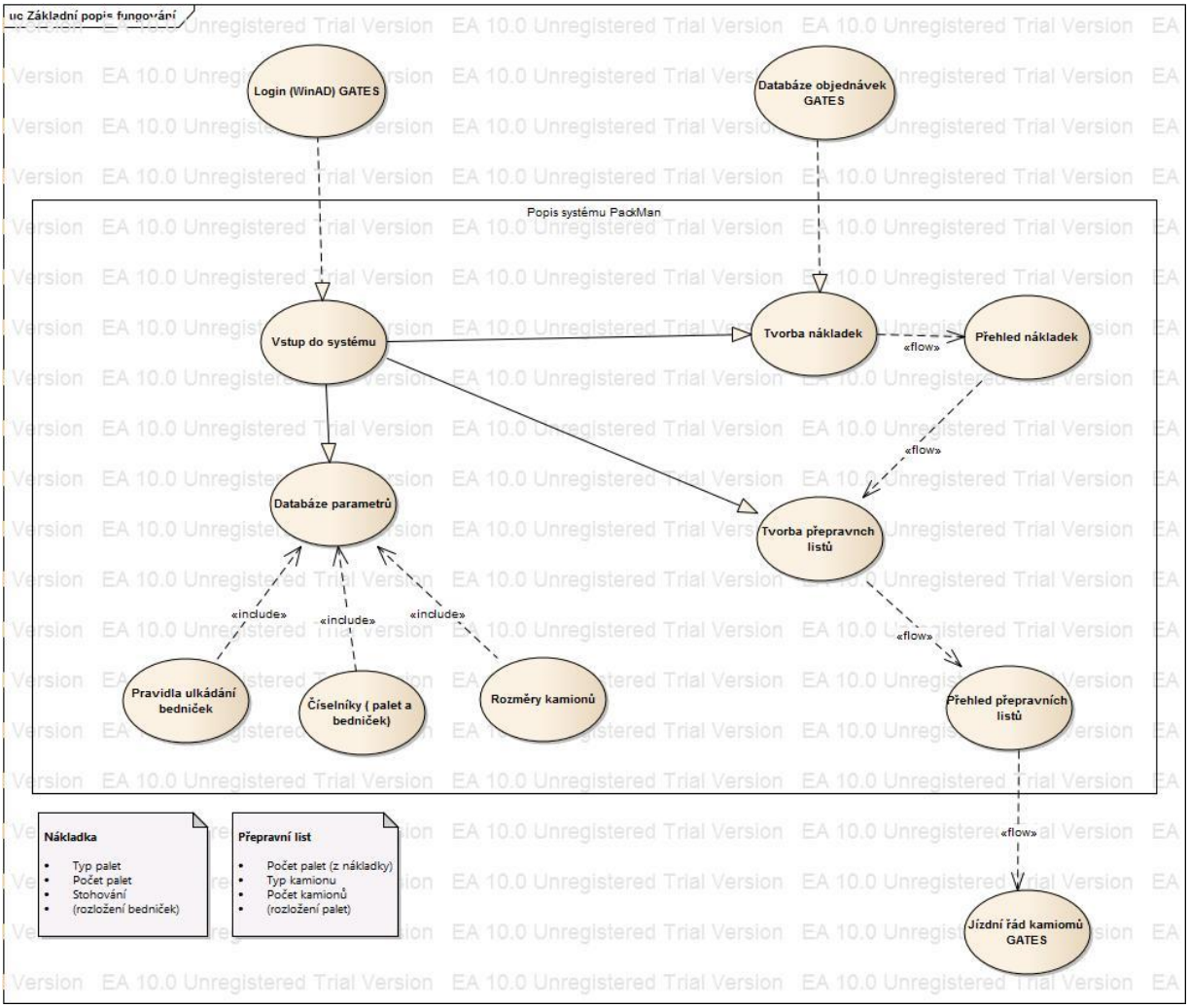
Má právo pouze zobrazit již dokončené nákladky a nákladní listy, je pro něj ale připraveno jiné uživatelské (dotykové) rozhraní

Dále byly pro potřeby popisu aktivit vytvořeny dva fiktivní uživatelé, a to uživatel (obecný) a systém. Role jednotlivých aktérů a jejich vztah ke konkrétním částem systému je znázorněn v následujícím grafu.

## Části systému a role

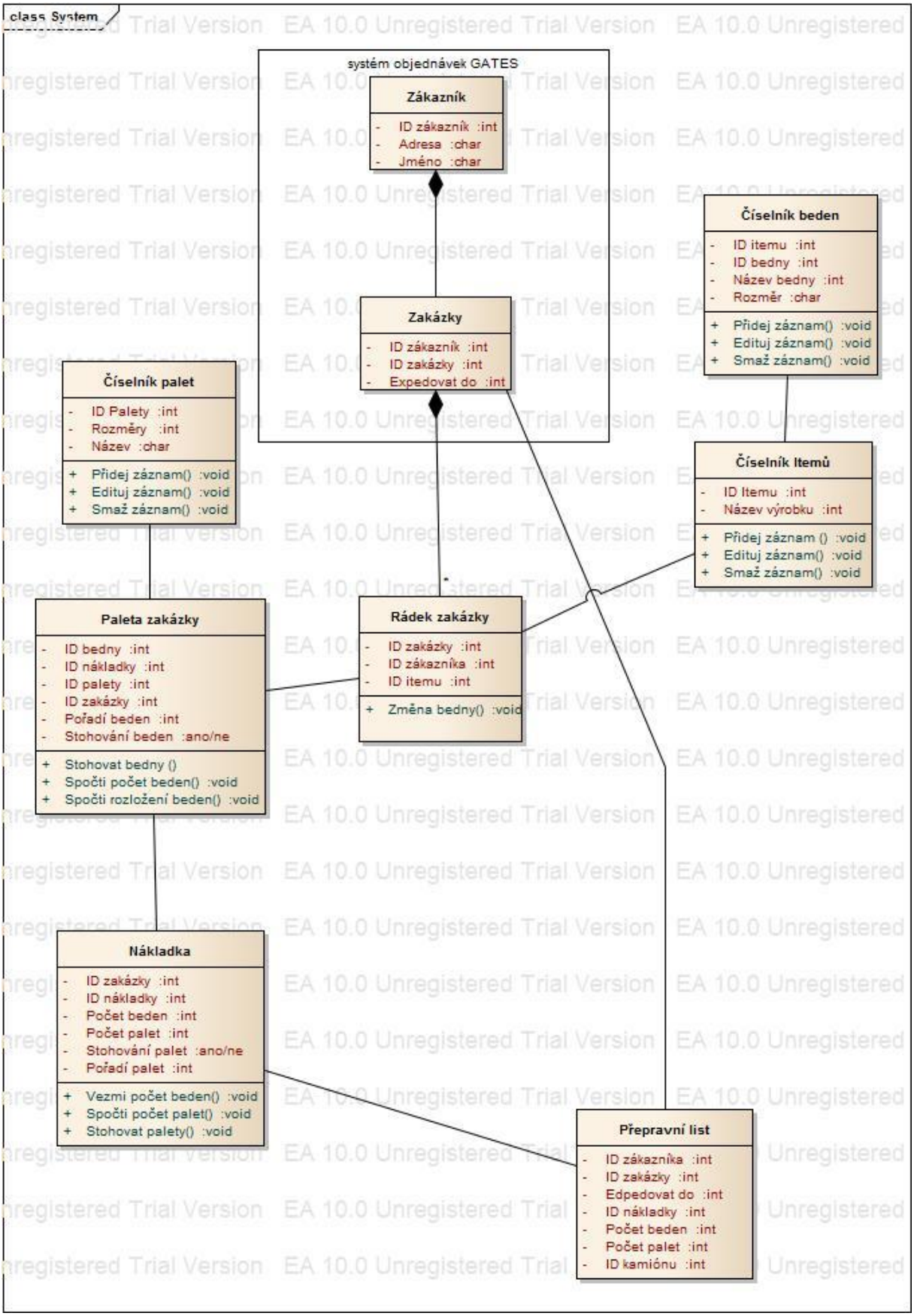


## Základní popis fungování



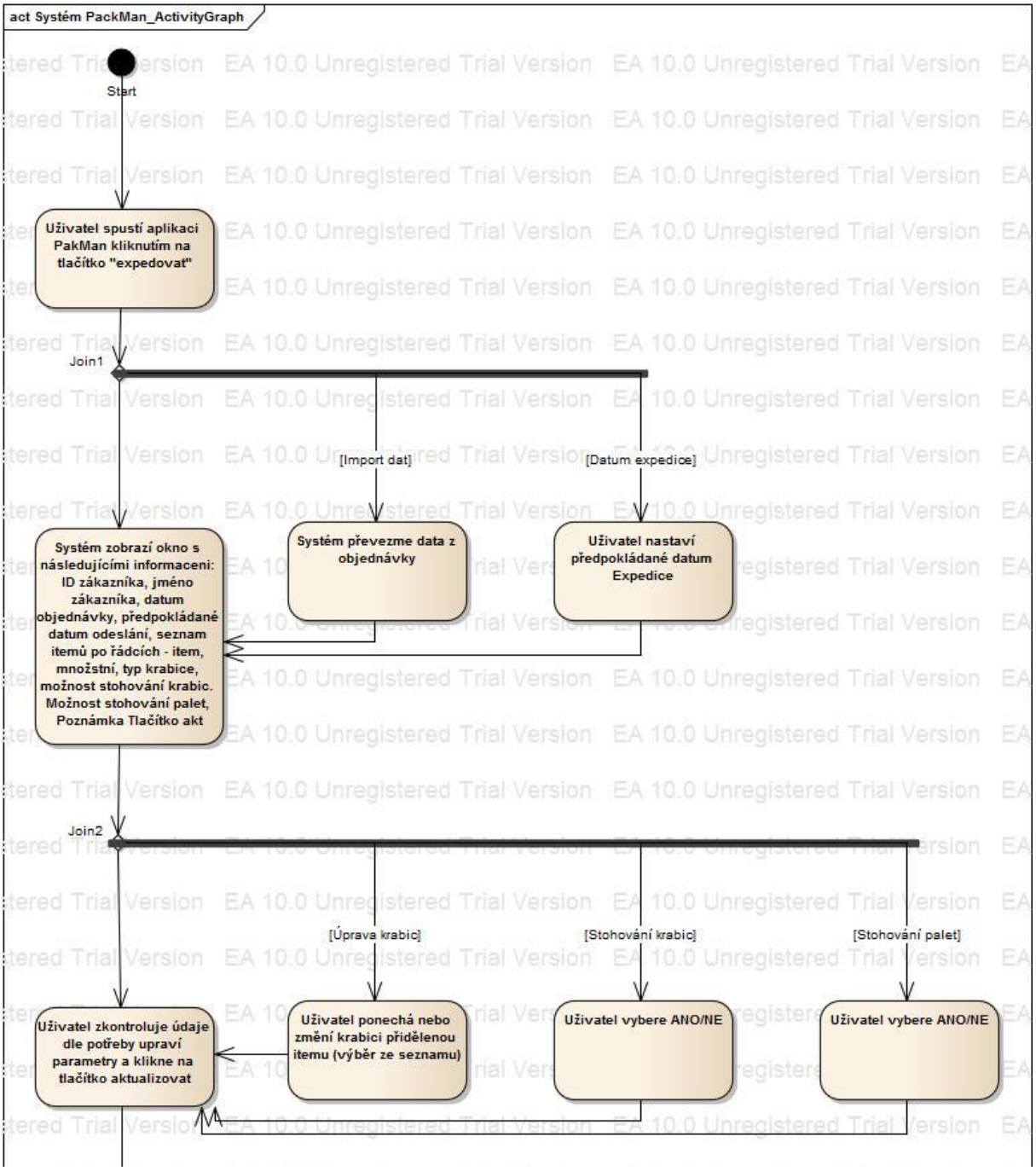

Tento diagram přehledným způsobem zobrazuje fungování systému a vnitřní toky dat. Toto bývá obvykle znázorněno v rámci jednoho use case digramu společně s určením aktérů a jejich rolí. Nicméně, pro přehlednost jsem se po domluvě s programátorem a zadavatelem, rozhodl dané vztahy znázornit takto ve dvou oddělených diagramech.

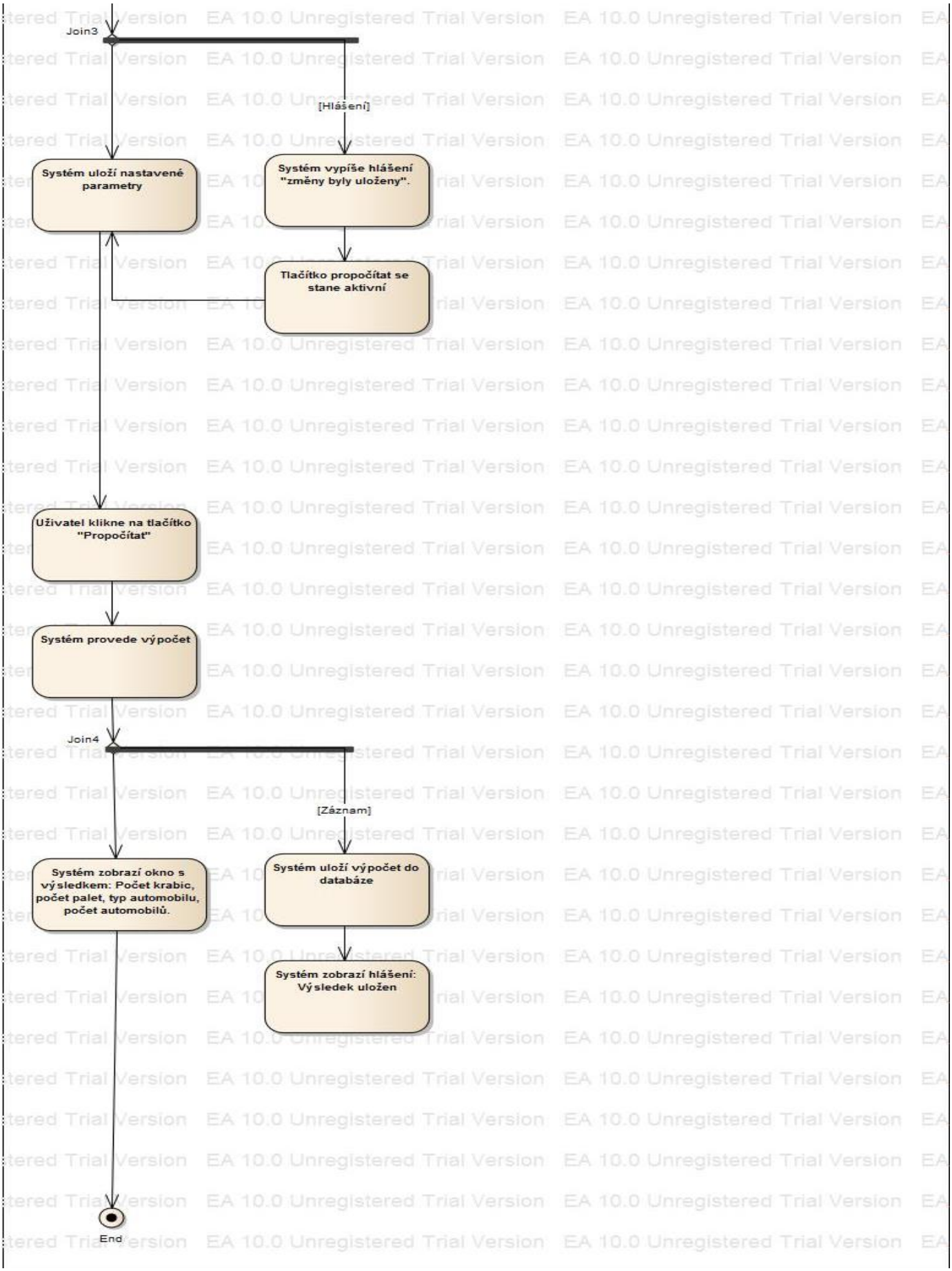
**Diagram tříd** Tento diagram znázorňuje jednotlivé třídy a vztahy mezi nimi, neboli způsob uložení dat v systému, vstup externích dat a metody pro práci s daty. Jedná se v této chvíli pouze o koncept, proto některé metody chybí.



## Diagram aktivit

Tento diagram znázorňuje aktivitu a interakci uživatele a systému. V tomto případě je znázorněno zpracování nové objednávky v systému, tedy vytvoření nové náklady a nákladního listu. Obvykle bývá doplněn o scénář. V tomto případě, na žádost zadavatele byl scénář přímo vepsán do jednotlivých částí diagramu.





**Resumé**

V tomto předmětu jsem se dověděl podstatnou část informací, potřebných ke zdárnému napsání diplomové práce a rovněž se mi podařilo vytvořit kvalitní koncept – základ pro další jednaní se společností a vývoj aplikace.

# ZÁVĚR

Tento učební text vznikl za podporu projektu INO_F – „Inovace studijních programů na Obchodně podnikatelské fakulty Slezské univerzity v Karviné.

Učební pomůcka je určena pro studijní program „Systémové inženýrství“, oboru Manažerská informatika, navazující stupeň studia 1. ročník. Vyžaduje předběžnou průpravu získanou v předmětech Databáze a souvisí s předmětem Objektové programování.

# SEZNAM POUŽITÉ LITERATURY

**[1] [Amb2004]** AMBLER, S. W. The Object Primer: Agile Model-Driven Development with UML 2.0. 3. edition. Cambridge University Press, 2004. 572 s. ISBN 0521540186.

**[2] [Arl2003]** ARLOW, J. NEUSTADT, I. UML a unifikovaný proces vývoje aplikací: Objektově orientovaná analýza a návrh prakticky. Přel. Bogdan Kizska. Brno: Computer Press, 2008. 387 s. ISBN 80-7226-947-X.

**[3] [Arl2008]** ARLOW, J. NEUSTADT, I. UML 2 a unifikovaný proces vývoje aplikací: Objektově orientovaná analýza a návrh prakticky. Přel. Bogdan Kizska. 1.vyd. Brno: Computer Press, 2008. 567 s. ISBN 978-80-251- 1503-9.

**[4] [Boo1998]** BOOCH, G. JACOBSON, I. RUMBAUGH, J. The Unified Modeling Language User Guide. 1.vyd. Addison Wesley, 1998. ISBN 0- 201-57168-4.

**[5] [Buch2002]** BUCHALCEVOVÁ, A. STANOVSKÁ, I. ŠÍMNEK, M. Základy softwarového inženýrství: základní témata. 1.vyd. Praha: Vysoká škola ekonomická, 2002. 198 s. ISBN 80-245-0346-8.

**[6] [Buch2007]** BUCHALCEVOVÁ, A. PAVLÍČKOVÁ, J. PAVLÍČEK, L. Základy softwarového inženýrství - materiály ke cvi4ení. 1.vyd. Praha: Vysoká škola ekonomická, 2007. 222 s. ISBN 987-80-245-1270-9.

**[7] [Fow2003]** FOWLER, M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. 3. edition. Addison Wesley, 2003. 208 s. ISBN 0- 321-19368-7.

**[8] [Kan2004]** KANISOVÁ, H. Miller, M. UML srozumitelně, 1.vyd, Brno: Computer Press, 2004. 158 s. ISBN 80-251-0231-9.

**[9] [Lar2001]** LARMAN, C. Applying UML and patterns : an introduction to object-oriented analysis and design and the unified process. 2.vyd. Upper Saddle River: Prentice Hall PTR, 2002. 627 s. ISBN 0-13-092569-1.

**[10] [Pen2003]** PENDER, Tom. UML Bible. 1.vyd. Indianapolis: Wiley Publishing, Inc., 2003. ISBN 0-7645-2604-9.

**[11] [Rum2004**] RUMBAUGH, J. JACOBSON, I. BOOCH, G. The Unified Modeling Language Reference Manual. 2.vyd. Addison-Wesley, 2004. ISBN 032124562853

**[12] [Schm2001]** SCHMULLER, J. Myslíme v jazyku UML : knihovna programátora. přel. Jiří Hynek. 1.vyd. Praha: Grada, 2001. 360 s. ISBN 80-247-0029-8.

**[13] [Str2007]** STŘÍŽOVÁ, V. HORNÝ, S. SVATÁ, V., VÁCLAVÍKOVÁ, M. Systémové pojetí (hospodářské) organizace. 1.vyd. Praha: Oeconomica, 2007. 239 s. ISBN 978-80-245-1265-5.

**[14] [Lef2010]** LEFFINGWELL, D. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise***.** Addison-Wesley Professional, 2010, 560 s., ISBN 0-321-63584-1

**[15] [Rat2011]** Rational Unified Process : Best Practices for Software Development Teams**,** [online]. Rational Software. [cit. 2. 4. 2011].

**[16] [Fow2009]** FOWLER, M**.** Destilované UML**.** Grada Publishing a.s. 2009. 176 s. ISBN 978-80-247-2062-3

**[17] [ALD2005]** ALDORF, F. Metodika RUP - diplomová práce

## 11.1 INTERNETOVÉ ZDROJE:

**[1]** http://mpavus.wz.cz

**[2]** http://uml.czweb.org/index.html

**[3]** http://www.rational.com

**[4]** http://www.ibm.com

**[5]** http://www.sparx.com

## 11.2 SEZNAM UKÁZKOVÝCH SEMINÁRNÍCH PRACÍ:

**[1] Slechan L.** Objednávka na e-shopu, OPF SU Karviná, ak. rok 2013/14

**[2] Dziadek L.** Import zboží do IS společnosti eShopSystem, OPF SU Karviná, ak. rok 2010/11

**[3] Janoušek M.** Registrace do eshopu a nákup motoru**,** OPF SU Karviná, ak. rok 2013/14

**[4] Cvinček P.** Poštové služby, OPF SU Karviná, ak. rok 2011/12

**[5] Stach V.** IS Svaz, OPF SU Karviná, ak. rok 2008/9

**[6] Dittrich J.** Návrh aplikace pro optimalizaci procesu vyskladnění, OPF SU Karviná, ak. rok 2013/14

# PŘÍLOHA Č. 1 SEZNAM OBRÁZKŮ

Obrázek 1: Strukturální pojetí - kód a data ..... 7
Obrázek 2: Historie vzniku UML ..... 15
Obrázek 3: Aktér = uživatelská role vůči systému ..... 21
Obrázek 4: Uživatel ve více rolích vůči systému ..... 22
Obrázek 5: Příklad scénáře případu užití ..... 22
Obrázek 6: Případ užití – Založení nového zákazníka ..... 23
Obrázek 7: Případ užití: Příjem spotřebiče do opravy ..... 24
Obrázek 8: Porovnání třídy analytického a návrhového modelu tříd ..... 27
Obrázek 9: Doménový model tříd ..... 29
Obrázek 10: Objektový diagram ..... 32
Obrázek 11 Prvky stavového diagramu ..... 33
Obrázek 12: Stavový diagram ..... 34
Obrázek 13: Prvky diagramu aktivit ..... 37
Obrázek 14: Použití plaveckých drah v diagramu aktivit (grafický zápis) ..... 38
Obrázek 15**:** Diagram aktivit ocenění ..... 39
Obrázek 16**:** Diagram aktivit autobazar – vytvoření smlouvy ..... 40
Obrázek 17: Sekvenční diagram popisující vytvoření ocenění vozidla ..... 43
Obrázek 18: Sekvenční diagram ocenění (grafické znázornění stereotypů) ..... 44
Obrázek 19: Týmová práce při vývoji v software IBM Rational development platform ..... 45
Obrázek 20: Příklad vývojového prostředí IBM Developer ptatform – USE CASE Model ... 46
Obrázek 21: Vývojové prostředí Enterprise Architect firmy Sparx – CLASS DIAGRAM .... 47
Obrázek 22: Schéma projektu dle metodiky RUP ..... 49
Obrázek 23: Iterativní vývoj ..... 51
Obrázek 24: Životní cyklus projektu ..... 53
Obrázek 25: Obvyklá náročnost jednotlivých fází (převzato z [RUP]) ..... 53
Obrázek 26**:** Symboly používané v RUP ..... 61
Obrázek 27: Tvorba podnikového modelu ..... 62
Obrázek 28: Správa požadavků ..... 64
Obrázek 29: Revize návrhu a architektury ..... 66
Obrázek 30: Implementace ..... 67
Obrázek 31: Testování ..... 68
Obrázek 32: Nasazení ..... 70
Obrázek 33: Řízení projektu ..... 71
Obrázek 34: Řízení změn a konfigurace ..... 73
Obrázek 35: Správa prostředí ..... 75
Obrázek 36: Diagram tříd ..... 91
Obrázek 37: Diagram případů užití ..... 92

# PŘÍLOHA Č. 2 STRUKTURA UML

*Rozdělení a přehled diagramů*

| Statická struktura | Dynamické chování | Správa modulů |
|---|---|---|
| diagram tříd (Class Diagram) | use case diagram | balíčky (Packages) |
| objektový diagram (Object Diagram) | sekvenční diagram (Sequence Digram) | subsystémy (Subsystems) |
| komponentový diagram (Component Diagram) | diagram činností (aktivit) (Activity Diagram) | modely (Models) |
| diagram nasazení (Deployment Diagram) | diagram spolupráce (Collaboration Diagram) | |
| | stavový diagram (Statechart Diagram) | |

*Role tvůrců a diagramy.*