UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE
FAKULTA PRÍRODNÝCH VIED

NÁVRH INFORMAČNÉHO SYSTÉMU PROSTRIEDKAMI DÁTOVÉHO A PROCESNÉHO MODELOVANIA

Bakalárska práca

2010 **Marek Daniš**

UNIVERZITA KONŠTANTÍNA FILOZOFA V NITRE
FAKULTA PRÍRODNÝCH VIED

NÁVRH INFORMAČNÉHO SYSTÉMU PROSTRIEDKAMI DÁTOVÉHO A PROCESNÉHO MODELOVANIA

Bakalárska práca

Študijný program: Aplikovaná informatika
Študijný odbor: 9.2.9. - Aplikovaná informatika
Školiace pracovisko: Katedra informatiky
Školiteľ: Mgr. Martin Drlík, PhD.

Nitra 2010 **Marek Daniš**

Poďakovanie

Touto cestou vyslovujem poďakovanie pánovi Mgr. Martinovi Drlíkovi, PhD. za pomoc, odborné vedenie, cenné rady a pripomienky pri vypracovaní mojej bakalárskej práce.

Abstrakt

DANIŠ, Marek: Návrh informačného systému prostriedkami dátového a procesného modelovania [Bakalárska práca]. Univerzita Konštantína Filozofa v Nitre, Fakulta prírodných vied, Katedra informatiky.
Školiteľ Mgr. Martin Drlík, PhD. Nitra 2010, 66 s.

Začiatok práce stručne popisuje pojmy softvérové inžinierstvo a informačný systém. Nasledujúce kapitoly sa zaoberajú vývojom metodík a vývojom modelov životných cyklov. V skratke sa stretneme aj so staršími modelmi. Väčšia pozornosť je venovaná prevratnému modelu životného cyklu – vodopádový model. A tak isto je i značná časť venovaná metodike RUP. V ďalšej časti sa práca venuje problematike procesného modelovania a popisu základných diagramov a symbolov. Následne práca oboznamuje i dátové modelovanie, kde nájdeme i logický a fyzický dátový model. Záver práce sa venuje jazyku UML a jeho využitiu počas analýzy a návrhu IS. Stretneme sa s pojmami ako požiadavky, use-case diagramy, slovné scenáre, ale aj s objektmi a triedami v jazyku UML. Potom sa oboznámime s históriou a základným popisom nástroja CASE. Vo finálnej časti práce nájdeme návrh pre informačný systém divadlo.

Kľúčové slová: informačný systém, procesné modelovanie, dátové modelovanie, UML, CASE

Abstract

DANIŠ, Marek: The design of information system by means of process and date modelling – tools. [Bachelor thesis]. Constantine the Philosopher University in Nitra, Faculty of Natural Sciences, Department of Computer Science.
Supervisor: Mgr. Martin Drlík, PhD. Nitra 2010, 66 p.

The beginning of the thesis shortly describes terms such as software engineering and information system. The following chapters deal with the development of the methodology and life cycle models. We will briefly touch the older models as well. Much more attention is devoted to the revolutionary life cycle model - also called waterfall model. At the same time, a major part handles with the RUP methodology. The part to come next copes with the issue of process modeling and the description of basic diagrams and symbols. The thesis further introduces data modeling, where logic and physical data model is to find. The last part of the thesis covers UML language and its use throughout the IS analysis and design. We will treat terms such as use-case diagrams, word scenarios and last but not least objects and classes in UML language. Then we will become familiar with the history and basic description of the CASE tool. In the final part of the thesis we will find the design for information system theatre.

Keywords: information system, process modeling, date modeling, UML, CASE

Obsah

Úvod

Ako ľudstvo sme museli prejsť dlhú cestu, kým sme od konských záprahov začali využívať stále dokonalejšie dopravné prostriedky. Takisto aj vo sfére počítačového sveta, sme prešli dlhým vývojom, pokým sme od diskiet, začali používať, dnes všetkým známe USB-kľúče. Vývoj počítačov, a s tým súvisiaci aj vývoj softvérov, má rýchle napredovanie, a softvér, s ktorým sme pracovali pred niekoľkými rokmi, sa dnes už považuje za zastaralý, ba v mnohých prípadoch v dnešných počítačoch nepoužiteľný.

V minulosti sa tvorba softvéru robila skôr formou pokusu a omylu. Dnes, však na základe týchto poznatkov, sme schopní vytvoriť pomerne stabilný softvér, ktorý môže slúžiť tak širokej verejnosti, ako aj vytvorenie zložitejších softvérových riešení pre užšiu skupinu používateľov. Pritom je veľmi dôležité, aby tento softvér pracoval správne, a stále sa čoraz viac zdokonaľoval.

Vytvorenie informačného systému je dlhodobý a komplikovaný proces. Už pri jeho návrhu sa môžeme stretnúť s rôznymi problémami. Preto je dobré riadiť sa určitou metodikou, vďaka ktorej môžeme znížiť riziko predčasného neúspešného ukončenia projektu. Pri návrhu informačného systému (IS) prostriedkami procesného a dátového modelovania sa používa jazyk UML. Pričom na uľahčenie práce sa používa CASE nástroj.

V súčasnosti je dôležité, aby sa softvér vyvíjal rýchlo, pričom sa musí vývoj prispôsobovať stále sa meniacim požiadavkám zákazníka, a s čo najväčšou efektivitou, tak aby sa takýto projekt neskončil krachom.

Ciele práce

Cieľom tejto práce je opísať problematiku a vývoj softvérového inžinierstva, informačného systému (IS) a jeho tvorbu pomocou dátového a procesného modelovania. V bakalárskej práci chceme na príklade informačného systému divadla poukázať na možnosti ich využitia.

Cieľom našej práce je navrhnúť taký informačný systém divadla, ktorý uľahčí prácu jednotlivým aktérom. IS má napomáhať pri príprave divadelnej hry už od jej začiatkov – dramaturgovi pri výbere režiséra, pri zostavovaní hereckého tímu pre konkrétne divadelné predstavenie. Pomôže pri zostavovaní jednotlivých nácvikov, predpremiéry, premiéry až po niekoľkonásobné hranie. Do tejto etapy má už prístup aj vonkajší element – divák, ktorý má možnosť si cez spomínaný IS rezervovať i objednávať lístky.

Návrh IS divadlo bude obsahovať funkčné a nefunkčné požiadavky, diagramy prípadov použitia, slovné scenáre, maticu sledovateľnosti požiadaviek, diagram analytických tried.

1 Softvérové inžinierstvo

Pre pochopenie metodiky je potrebné najprv popísať softvérové inžinierstvo.

Softvérové inžinierstvo je súhrn pojmov, postupov a činností, ktoré by mali zaistiť efektívnu tvorbu softvéru. Nezaoberá sa len samotným vývojom, ale zahrňuje aj ďalšie oblasti ako komunikácia so zákazníkom, softvérovou etikou a podobne.

Definícia : Fritz Bauer (Konferencia NATO, 1968): "Softvérové inžinierstvo je zavedenie a používanie riadnych inžinierskych princípov tak, aby sa dosiahli ekonomické tvorby softvéru, ktorý je spoľahlivý a pracuje účinne na dostupných výpočtových prostriedkoch" [1].

V tejto definícii je skryté všetko podstatné čím sa zaoberá a o čo sa softvérové inžinierstvo snaží. Teda úlohou programátora (prípadne softvérovej firmy) nie je len napísať program, ale jeho práca musí zahrňovať viacero ďalších nemenej dôležitých aspektov, ktorých súhrn je popisovaný práve softvérovým inžinierstvom [1].

1.1 Úspešné podmienky ekonomickej tvorby softvéru

Václav Kadlec (2004) vo svojej publikácii „Agilní programování" uvádza niekoľko príkladov podmienok pre úspešnú ekonomickú tvorbu softvéru:

- Vhodne zostaviť vývojový tím – nesmie mať ani málo, ani veľa členov. Pričom by tento tím mal pokrývať všetky role. Mal by rešpektovať používanú metodiku a vhodný *truck faktor* (t.j. aby sa nestalo, že po odchode jedného člena nebude už danej problematike rozumieť nik iný a tým pádom nastane krach projektu) a mal by byť zárukou efektívnej práce ľudí. Efektivita práce ľudí je jedným zo základných pilierov ekonomickej tvorby softvéru.
- Správne zvoliť vývojový nastroj, správna voľba vývojového nástroja a prostredia podľa vnútorných podmienok, vybavenia a znalosti tímu môže ušetriť mnoho prostriedkov.
- Úvaha vyvinúť/kúpiť, treba sa zamyslieť nad tým či je výhodné vykonávať celý vývoj vlastnými silami, alebo zakúpiť systém v prípade ak už daný systém niekto vyvinul. Pretože pri odkúpení, preskúmaní a integrácii systému sa dá ušetriť veľké percento nákladov, oproti nákladom na vývoj nového systému.

- Nájsť spoločnú reč so zadávateľom, ak si firma a zadávateľ nerozumejú, tak v úvodnej fáze nemusí firma získať kontrakt, v neskorších fázach môže zas zadávateľ odmietnuť prevzatie.

Takýchto podmienok by sme mohli nájsť omnoho viac a všetky môžeme zahrnúť do pojmu „úspešná a ekonomická tvorba softvéru“, pretože zanedbanie akéhokoľvek bodu vedie k ekonomickým stratám [1].

2 Informačný systém

Informačný systém je súbor ľudí, technických prostriedkov, metód a činností, zabezpečujúcich zber, prenos, uloženie, výber, distribúciu a spracovanie dát (informácií) za účelom tvorby a prezentácie informácií pre potreby používateľov činných v systémoch riadenia. Teda je to súhrn technológií napomáhajúcich konkrétnej organizácii pri ich realizácii podnikovej stratégie [2].

Údaje v informačných systémoch musia byť zobraziteľné v zrozumiteľnej forme. Pričom údaje, ktoré daný systém poskytuje, sa pre rôznych aktérov zobrazujú v inej forme, a teda, inak zobrazíme informácie pre riaditeľa, návrhára a napríklad skladníka.

Vzhľadom na to, že nástroje informačného systému podporujú len určité činnosti pre danú firmu či organizáciu, nie je možné takýto systém zakúpiť ako iný softvér, ale treba navrhnúť a vytvoriť systém nový, prípadne upraviť už existujúci. Návrh informačného systému sa riadi konkrétnou metodikou [3].

3 Metodika a životný cyklus

Metodika a životný cyklus patria k najdôležitejším produktom softvérového inžinierstva. Rovnako ako má históriu celý tento odbor, prechádzali rôznymi obdobiami aj metodika a životný cyklus. Aj keď metodika i životný cyklus sa pokúšali prispôsobiť vývoj softvéru konkrétnym požiadavkám v danej dobe, existujú medzi nimi značné rozdiely. Model životného cyklu popisuje len „život“ aplikácie a postupnosť krokov pri jeho vývoji. Metodika väčšinou obsahuje aj radu ďalších informácií a predpisuje čo sa má v ktorom okamihu urobiť [1].

3.1 Metodika

Metodika tvorby informačných systémov je súhrnom etáp, prístupu, postupov, pravidiel, riadenia, metód, techník a nástrojov, ktoré pokrývajú celý životný cyklus IS. Skrývajú sa pod ňou všetky etapy riešenia. Metodika by sa mala zaoberať všetkými prvkami IS (pracovníkmi, procedúry, dáta, SW, HW atď.) a hľadá odpovede na otázky: kto?, kedy?, čo? a prečo?. Podrobnejšími otázkami (ako danú operáciu vykonať) sa metodika zaoberať nemusí. Taktiež napomáha tomu, aby sa uskutočnili činnosti pri tvorbe informačného systému v správnej časovej postupnosti, ako aj k dobrej organizácii pri práci na projekte, vytvoreniu dokumentácie a optimalizácii spotreby zdrojov pri tvorbe i prevádzke informačného systému [1,4].

Aby sa mohla metodika efektívne využiť musí spĺňať niekoľko nasledujúcich základných požiadaviek [3]:

- Jasná deklarácia súboru hodnôt, na ktorých je metodika založená.
- Určiť postup riešenia tak, aby bolo možné plánovať celý proces vývoja.
- Dôraz na všetky faktory a dimenzie, ktoré ovplyvňujú tvorbu a prevádzku informačného systému.
- Určenie priority riešenia.
- Odporúčanie metód, techník a nástrojov, ktoré je vhodné využiť v určitých fázach vývoja.

Metóda je nejaký konkrétny postup, ktorý vedie k vyriešeniu určitého čiastkového problému [1]. Na úrovni metód môžeme rozlíšiť rôzne prístupy k tvorbe informačného systému (napríklad funkčný, dátový alebo objektový prístup). Každá metóda potom rieši činnosti v určitej časti procesu vývoja informačného systému.

Príklady metód :

- Informačná analýza.
- Riadenie projektu.
- Štruktúrovaná funkčná analýza [4].

Technika vyjadruje ako sa dopracovať k požadovanému výsledku. Určuje presný postup jednotlivých činností a vyhradzuje pre ne presné pravidlá. Ďalej určuje spôsob použitia nástrojov, varianty rozhodnutí pri určitých situáciách. Technika je často úzko spojená s konkrétnym nástrojom. A aj predurčuje spôsob uvažovania či vyjadrovania. Technika je na rozdiel od metódy presnejšia v záveroch, ale obmedzenejšia v rámci okruhov použitia. Je špecializovanejšia na konkrétny problém.

Príklady techník [3,4]:

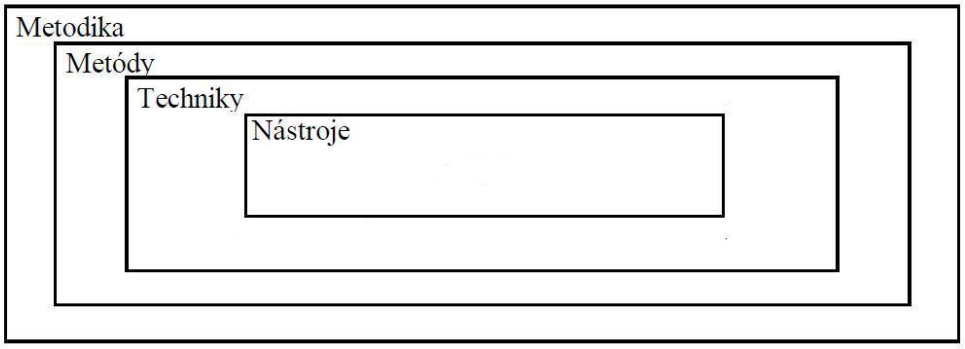
- Transakčná analýza.
- Normalizácia dátového modelu.
- Prototypovanie.
- Vedenie interview.
- Normalizácia údajov.
- Štruktúrované programovanie.
- Dátové modelovanie.

Nástroj je vyjadrovacím prostriedkom, ktorý slúži k uskutočneniu či namodelovaniu určitej činnosti v procese vývoja a prevádzkovania informačného systému a prostriedkom k vyjadreniu výsledku tejto činnosti. Je pomôckou pre techniky a často je spojený s konkrétnou technikou. Nástroje formalizujú vyjadrenie, preto je žiaduce aby boli čo najviac automatizované [4].

Príklady nástrojov [4]:

- DFD - diagram dátových tokov.
- ERD - diagram entít a ich vzájomných vzťahov.
- SD – štruktúrny diagram.

Hierarchia vzťahov vyplývajúcich z hore uvedeného popisu je nasledovná[4]:
Metodika – Metóda – Technika – Nástroj



Obr. 1 – Hierarchia Metodika – Metóda – Technika – Nastroj [25].

Tieto závislosti nie sú tak jasné, ako by sa mohlo na prvý pohľad zdať. Určitý nástroj nemusí byť nutne viazaný na jedinú techniku, ale môže byť natoľko všeobecný, že môže byť použitý viacerými metódami, prípadne môže nastať opak, teda že bude tak špecializovaný, že ho bude môcť využiť výhradne jediná metóda. To isté platí i o technikách a metódach v rámci metodík. Teda, že techniky a metódy nemusia patriť jednoznačne konkrétnej jedinej metodike, ale viacerým, prípadne naopak, že bude patriť výlučne jedinej metodike. Vzťahy medzi metódami, technikami a nástrojmi, ako aj ich príslušnosť k metodikám, môžu byť rozmanité [3, 4].

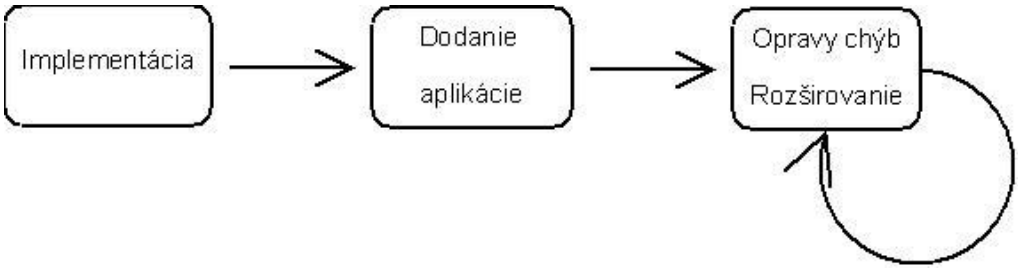
Budovanie informačných systémov a tvorba projektov v tejto oblasti prešla pomerne dlhým obdobím vývoja, v rámci ktorého boli vypracované metodologické postupy, overené príslušné metódy návrhu a aplikované príslušné nástroje pri návrhu IS. Metodika návrhu predstavuje súhrn postupov a čiastočných metód, ktoré definujú čo sa robí v procese tvorby IS. Určujú kedy a kým sa dané úkony vykonávajú, za použitia akých metód a nástrojov. Jedná sa vlastne o akúsi filozofiu tvorby a návrhu IS. Metódy návrhu určujú čo je potrebné vykonať v rámci konkrétnej fázy návrhu a aké prístupy je potrebné pritom dodržať [25].

Metodiky vývoja aplikácií patria medzi najdôležitejšie produkty softvérového inžinierstva, majú svoju históriu a prechádzali určitým vývojom. Metodiky sa snažili prispôsobovať k požiadavkám kladených na vývoj softvéru v danej dobe. Napríklad v čase softvérovej krízy sa snažili klásť veľký dôraz na fázu analýzy a špecifikácie, pretože práve chyby v týchto činnostiach viedli k danej softvérovej kríze. Keďže metodiky v súčasnosti kladú najväčšie požiadavky na to, aby bol výsledný softvér dodaný čo najskôr, tak sa

v dnešnej dobe používajú agilné metodiky. Existuje viacero metodík, pričom za najstaršiu sa považuje Model napíš a oprav [1].

3.1.1 Model napíš a oprav

Používal sa v začiatkoch vývoja programov. V týchto dobách, ešte neboli známe žiadne metodiky vývoja softvéru. Softvér písaný štýlom napíš a oprav (*Code and Fix Model*, prípadne *Build and Fix model*). Programátori v tých dňoch vtedy väčšinou programovali program len pre seba, aby im pomohol v ich práci. Ako sám názov vraví bolo to bez akýchkoľvek príprav, len sadli k PC a programovali. No ak sa počas práce s programom vyskytli nejaké problémy alebo chyby, tak skúsili daný problém či chybu nájsť a opraviť a takto stále dokola. V prípade takéhoto prístupu ani nemôžeme o nejakej metodike vravieť. Model i jeho názov vznikli spontánne. Tento model a softvér, ktorý bol na jeho základe vyvíjaný, boli neefektívne. No v dobe, kedy sa tento model používal, programy, ktoré vznikali boli malé (rozsahovo), takže tento model bol pre ne čiastočne použiteľný [1,5].



Obr. 2 – Schéma modelu napíš a oprav (Code and Fix model) [1].

3.1.2 Agilné metodiky

V súčasnosti sú najvhodnejšie tzv. agilné metodiky vývoja softvéru. Agilné metodiky sa vyznačujú niekoľkými základnými charakteristikami. Možno spomenúť iteratívny a inkrementálny vývoj s veľmi krátkymi iteráciami, dôraz na priamu osobnú komunikáciu v tíme, nepretržitú úzku spolupráca so zadávateľom, resp. používateľom, či rigorózne, opakované a priebežné automatizované testovanie [21].

V rámci svojej knihy Mike Cohn pekne zhrnul dôvody čoho sa máme držať pri plánovaní v agilných projektoch [22]:

- Plánovať treba často.
- Odhady veľkosti a času potrebných na vykonanie úlohy sa vykonávajú oddelene.
- Plánovanie sa robí na rôznych úrovniach.

- Plány sú založené na vlastnostiach produktu a nie úlohách.
- Malé príbehy pomáhajú s pracovným tempom.
- Rozrobené úlohy sú eliminované s každou iteráciou.
- Monitorovanie sa deje na tímovej úrovni.
- S neistotou sa počíta a plánuje sa s ňou.

Základným pilierom agilných metodík je manifest z roku 2001, ten vychádza z dvoch hlavných úvah [1]:

1. Prijať a umožniť zmenu je omnoho efektívnejšie, ako pokúšať sa jej zabrániť.
2. Je treba byť pripravený reagovať na nepredvídateľné udalosti, pretože tie určite nastanú: tento bod je možné formulovať i tak, že jedinou istotou je zmena.

Z týchto dôvodov dávajú autori prednosť [1]:

- individualitám a komunikácii pred procesmi a nástrojmi,
- prevádzky schopného softvéru pred obsiahlymi, objemnými dokumentáciami,
- spolupráci so zákazníkom pred uzatváraním veľa zmlúv,
- reakcii na zmenu pred striktným plnením plánu.

V súčasnosti sa stále zvyšuje počet konkrétnych metodík, ktoré bezprostredne vychádzajú z agilného manifestu, najvýznamnejší zástupcovia sú [1]:

- adaptívny vývoj softvéru,
- vlastnosťami riadený vývoj,
- extrémne programovanie,
- Lean Development,
- SCRUM Development Process,
- Crystal metodiky,
- Dynamic System Development Method,
- testami riadený vývoj.

3.2 Životný cyklus

Životný cyklus informačného systému predstavuje jednotlivé etapy procesu návrhu, implementácie a používania informačného systému. Pod pojmom model životného cyklu rozumieme formalizovaný popis činností a aktivít s definovanými väzbami ako aj časovou postupnosťou. Model životného cyklu pozostáva z jednotlivých procesov plánovania, analýzy a implementácie, ktoré sú bližšie popísané v jednotlivých kapitolách [7].

3.2.1 Proces plánovania

Životný cyklus informačného systému väčšinou začína systémovým výskumom, ale životný cyklus systému začína už pri procese strategického plánovania. Toto plánovanie musí byť v plnom súlade so strategickou víziou a strategickými plánmi rozvoja firmy. Pri procese plánovania majú vývojové a riadiace funkcie a aj metódy značný vplyv skrz spätnú väzbu na proces plánovania [7].

3.2.2 Proces analýzy

Výsledkom strategického plánovania informačného systému je zoznam požiadaviek, ktoré tvoria podklad pre proces analýzy.

Zdroje informácií môžu byť:

- interview s manažérmi,
- interview s koncovými používateľmi,
- dotazníkové prieskumy,
- výskum dokumentov, predpisov, tlačív, správ, manuálov, atď.,
- aktívny výskum procesov, činností a aktivít za účasti koncového používateľa.

Ako výsledok analýzy je pre nás zoznam potrieb a očakávaní používateľov, ten je vstupom pre proces špecifikácie riešenia potrieb. Z tohto procesu dostávame tzv. úvodnú štúdiu.

Úvodná štúdia obsahuje :

- špecifikáciu problému,
- základné požiadavky,
- cieľ projektu,
- špecifikáciu a hrubý logický návrh,
- varianty riešenia,
- analýza nákladov a prínosov,
- harmonogram riešenia,
- finančné náklady – rozpočet.

Úvodná štúdia je hlavným východiskom pre proces analýzy a logického návrhu systému. Výsledkom procesu návrhu je analýza a logický návrh procesov. Logický návrh obsahuje návrh technického, aplikačného a aj personálneho zabezpečenia informačného systému. Po schválení logický návrh postupuje do procesu implementácie [7].

3.2.3 Proces implementácie

Implementácia predstavuje vygenerovanie alebo vytvorenie programu (alebo programov), napísanie potrebnej dokumentácie, špecifikáciu nárokov na údaje a pod.. Cieľom je vytvoriť fungujúci systém, ktorý je realizáciou návrhu z predchádzajúcich etáp životného cyklu informačného systému. Taktiež sa uskutočňuje testovanie systému. Systém musí fungovať bezchybne a všetky stanovené požiadavky z predchádzajúcich fáz musia byť implementované. Musí byť vytvorená dokumentácia a popis pracovných procesov. Celá implementácia je podmienená predovšetkým spokojnosťou koncového používateľa. Po úspešnom procese nasadenia informačného systému nasleduje prevádzka a údržba, prípadne zlepšovanie informačného systému [7, 8].

Vývoj softvéru nie vždy spadá do kategórie „ideálnych procesov". Každému vyhovujú iné pravidlá, niekomu dokonca žiadne. Ťažko však hovoriť o pravom a univerzálnom spôsobe, pretože rôzne techniky majú svoje výhody, ktoré môžu byť v špecifických prípadoch nevýhodami a opačne. Medzi prvé modely životného cyklu patrí Stagewise model (striktná postupnosť fáz), ktorý bol definovaný v roku 1957. Avšak za najviac prevratný je považovaný vodopádovy model, ktorý vznikol v roku 1970 [1, 6].

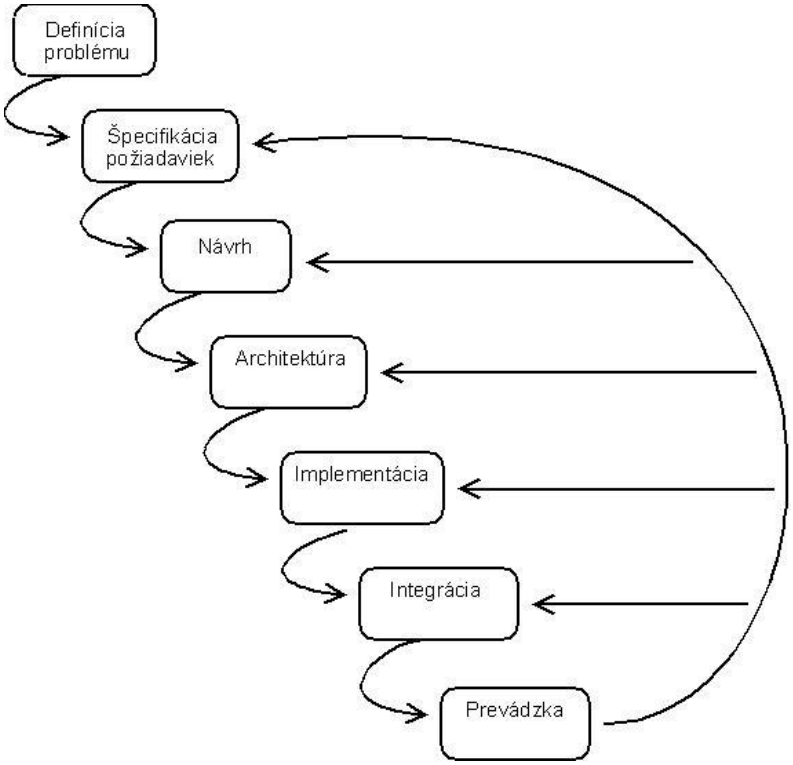
3.2.4 Striktná postupnosť fáz

Model striktná postupnosť fáz (stagewise model) bol definovaný už v roku 1957. Tento model je založený, ako vraví už sám názov, na striktnej postupnosti fáz, teda rozdeľuje vývoj softvéru na fázy :

- definícia problému,
- špecifikácia požiadaviek,
- návrh,
- architektúra,
- implementácia,
- integrácia,
- prevádzka.

Hlavným problémom tohto modelu je to, že neobsahuje žiadnu spätnú väzbu. Po skončení akejkoľvek fázy neprebiehala žiadna kontrola, nehodnotili sa doterajšie výsledky, nepreskúmavali sa požiadavky a ani sa nehľadali žiadne riziká. V tomto modeli vývoj postupoval len priamo dopredu. Jediné možné vrátenie sa do predchádzajúcej fázy bolo až

na konci samotného záveru – revalidácia. V rámci revalidácie je možné rozhodnúť o vrátení sa späť a o opakovanom vstúpení do procesu [1].



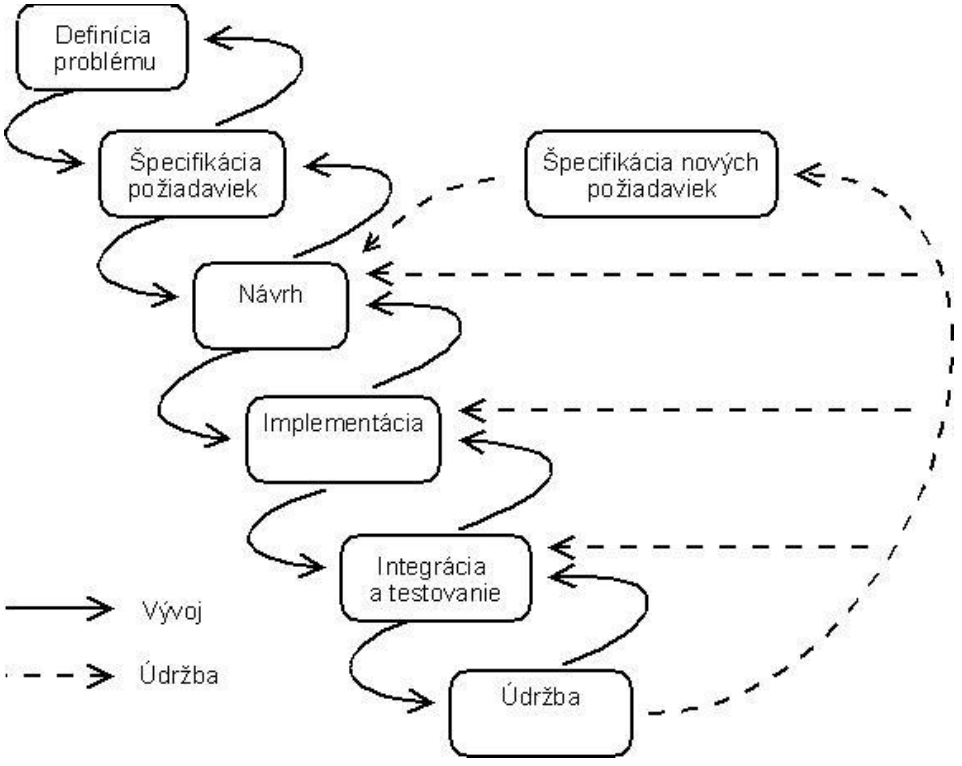
Obr. 3 – Postupnosť fáz v modeli stagewise [1].

4 Vodopádový model

Vodopádový model nie je metodikou vývoja softvéru, ale je to model životného cyklu. Tento model vznikol v roku 1970, v dnešnej dobe sa už používa stále menej a menej, pretože nie je vhodný pre rozsiahle projekty. Ale tento model prišiel ako prvý so základným členením softvérového procesu na jednotlivé aktivity a ich logickou postupnosťou. V tomto modeli sú všetky podstatné fázy vykonávané v predpísanom poradí, pričom ich iterácie sú minimálne alebo žiadne. Charakteristickým znakom je sekvencia jednotlivých fáz. Až po skončení jednej fázy sa začne fáza druhá, teda nebudú sa nikdy vykonávať dve fázy naraz. Ďalšou charakteristikou je striktná postupnosť krokov a fáz.

Fázy vodopádového modelu [1]:

1. definícia problému, poznanie zákazníka, preniknutie do cieľovej oblasti.
2. analýza a špecifikácia požiadaviek.
3. návrh systému.
4. implementácia systému.
5. integrácia a testovanie systému.
6. prevádzka a údržba.



Obr. 4 – Základná štruktúra vodopádového modelu životného cyklu [1].

Pokiaľ pracujeme na prvých šiestich fázach, teda od definície požiadaviek až po testovanie vytvoreného systému, tak sa nachádzame v kategórii definícia a vývoj. V tejto časti sa vytvára návrh, implementácia, testové prípady a všetky ďalšie dôležité náležitosti. Tie sú vypracovávané vývojovým tímom v spolupráci so zákazníkom. V týchto fázach postupujeme po vodopádovom modeli vždy len o jednu úroveň dopredu, prípadne späť. Ak nastanú nejaké komplikácie v určitej fáze je možné vrátiť sa späť a vykonať potrebné zmeny. Po prevedení týchto úprav musia všetky inovované dokumenty opäť prejsť schvaľovacím procesom. Toto je jediný znak flexibility pri tomto modely. Žiadne iné zmeny nie sú možné. Až po vyhotovení celého produktu a skončení fázy testovania, odovzdávame tento produkt zákazníkovi. A v tejto poslednej fáze sa môžeme vrátiť do hociktorej predošlej fázy a upraviť ju.

Tento model bol v minulosti pre svoju dobu postačujúci, pretože v tých časoch nik nerozmýšľal nad tým, že život softvérového produktu nekončí pri jeho odovzdaní. Vo vodopádovom modeli je pre vývojový tím zákazník potrebný len pri analýze a špecifikácii požiadaviek a potom až vo fáze údržby. Medzi týmito fázami nie je vývojový tím tlačený do komunikácie so zákazníkom [1].

Vodopádový model sa stále rozvíja a rôzni autori ho vylepšujú a upravujú, preto existuje mnoho variantov tohto modelu. Väčšina sa líši prakticky len rozdelením na etapy. Napríklad pôvodný model podľa Winsotna W. Royca obsahuje sedem fáz v nasledujúcom poradí : špecifikácia požiadaviek, návrh, implementácia, integrácia, testovanie a ladenie, inštalácia a údržba. Niektoré iné variácie tohto modelu majú len päť fáz, pretože niektoré fázy sú zlúčené [1,23,24].

V tejto práci, sa ale budeme ďalej zaoberať modelom, ktorý rozvíja v svojej knihe Václav Kadlec.

4.1 Fáza definície problému

Hlavnou úlohou tejto fázy je pochopiť zámer zákazníka. Prekonzultovať s ním čo najviac a zistiť prečo daný systém potrebuje, v čom mu má uľahčiť prácu, aké má na systém požiadavky a čo od systému očakáva. Cieľom je „zmapovať terén“. Výsledkom tejto fázy by mal byť dokument „Úvodná štúdia“, kde sú zhrnuté všetky informácie. Úvodná štúdia je zároveň hrubý a základný popis požiadavkou na systém [1].

4.2 Fáza analýzy a špecifikácia požiadaviek

V tejto fáze analýzy skúmame, čo má presne daný systém robiť a ako by to mal robiť. Popisujeme tu požiadavky na systém exaktne, používaním presných termínov, pomocou presných formulácií a vyčísliteľných charakteristík. Treba tu pochopiť presne problém zákazníka, ktorý má daný systém riešiť. Výsledkom analýzy je „Špecifikácia požiadaviek“. Na tento dokument sú kladené vysoké nároky, na jeho štruktúru, formu, obsah i spôsob vytvárania [1].

V rámci fázy špecifikácie požiadaviek riešime, čo bude daný systém robiť, no nie to ako bude implementovaný. Cieľom dokumentu „Špecifikácia požiadaviek“, je popísať aplikáciu tak aby tomuto popisu zákazník rozumel (popis bude písaný jazykom zákazníka). Daný dokument by mal byť odsúhlasený a podpísaný zákazníkom. Pri vodopádovom modeli predpokladáme , že po analýze bude dokončená špecifikácia požiadaviek a tá sa už bude meniť až do odovzdania aplikácie (tento predpoklad ale v praxi nemusí byť pravdivý) [1].

4.3 Fáza návrhu a vytvárania architektúry

Hlavným cieľom tejto fázy je prejsť špecifikáciu požiadaviek a podľa nej navrhnúť pre daný projekt najvýhodnejšiu architektúru systému. Výstupom tejto fázy je kompletná architektúra systému, rozbor a popis modulov a ich význam, spôsob komunikácie modulov, rozdelenie aplikácie na podprogramy, popis získavania a uchovávania informácií a definitívne stanovenie technológii.

Václav Kadlec (2004) vo svojej publikácii „Agilní programování“ uvádza nasledovný postup návrhu architektúry [1]:

1. určenie implementačného prostredia, vývojového nástroja, programovacieho jazyka, všetkých potrebných technológií.
2. vytvorenie architektúry systému, logické rozdelenie systému na subsystémy a ďalšie funkčné celky.
3. výber a návrh implementačných modulov, „na mapovanie“ logického návrhu do fyzickej – implementačnej – štruktúry.
4. definícia chovania modulov, špecifikácia práce s dátami (uchovávanie, formát a pod.).
5. usporiadanie výsledkov, diskusia, spresnenie architektúry.

K návrhu môžeme pristupovať štruktúrovane alebo objektovo. Ako náhle poskytneme programátorom špecifikáciu požiadaviek (z fázy 2) a popis návrhu(z fázy návrhu), mali by byť schopní implementovať celý systém bez ďalších impulzov. Pre návrh existuje niekoľko konkrétnych metód (štrukturálne metódy – funkčná dekompozícia, modulárne programovanie; objektové metódy – Rational Unified Process (RUP), OOSE, Booch a pod.) [1].

4.4 Fáza implementácie

V tejto fáze má za úlohu vývojový tím naprogramovať aplikáciu, tak aby bol zákazník spokojný. V ideálnom prípade by programátori mali mať jasno, na základe predchádzajúcich dokumentov, čo majú robiť. Z hľadiska metodiky nie je dôležite aký jazyk použijú k implementácii [1].

4.5 Fáza integrácie a testovania

Cieľom tejto fázy je, aby aplikácia fungovala správne, a aby vykonávala to čo zákazník vyžaduje. Ideálne je, keď vývojový tím otestuje každý kúsok zdrojového kódu, každú podmienku a všetky kombinácie možných vstupov. Existuje veľa metód testovania [1].

4.6 Fáza prevádzky a údržby

Po úspešnom prevzatí aplikácie zákazníkom, nastáva nikdy nekončiaci proces úprav, opráv, zlepšovaní a dolaďovania aplikácie [1].

5 Metodika Rational Unified Process (RUP)

RUP je rozsiahla, prepracovaná, objektovo orientovaná iteratívna metodika vývoja softvéru. Táto metodika bola vyvinutá firmou Rational Software (v roku 1995 odkúpená firmou IBM), ktorá prirovnáva svoj produkt k on-line inštruktorovi, ktorý poskytuje metodické pokyny, inštrukcie, šablóny a príklady pre aspekty a fázy vývoja softvéru. RUP je výrazne objektovo orientovaná metodika, ako jej základný element je prípad použitia (Use Case). Metodika RUP rozpracováva podrobne iterácie kde určuje: kto to má robiť (roles), ako to má robiť(activities) a čo bude výsledkom(artifacts) [1,9].

5.1 Najlepšie praktiky (Best Practices)

Metodika RUP definuje 6 takzvaných najlepších praktík (best practices) , čo sú praktiky užitočné pri realizácii vývoja softvéru, založené na dlhodobých skúsenostiach v tejto oblasti. Pri svojom použití majú zaistiť efektívnejší výkon, prepracovanejší režim kvality a lepšie výsledky. Praktiky sú v rámci metodiky podporované rôznymi nástrojmi pre automatizáciu špecifických procesov vývoja softvéru.

Medzi tieto praktiky patria :

- Iteratívny vývoj - táto praktika odstraňuje problém tlačenia rizika pred sebou, ktoré nastáva pri tradičnom sekvenčnom prístupe. Iteratívny vývoj softvéru vychádza zo špirálového modelu kde sa riziká hľadajú priebežne. Takýto prístup delí projekt do časových sekov (iterácií), pritom by pri skončení každej iterácie mala vzniknúť spustiteľná verzia. Ďalšie výhody sú napr. ľahšia správa zmien, dobrá znovupoužiteľnosť a dokonalejší pristúp k zákazníkovi.
- Správa požiadaviek – keďže požiadavky sa môžu často meniť, tak cieľom správy požiadaviek v metodike RUP je aktívny zber a dokumentácia požiadaviek, prebiehajúci medzi zadávateľom a dodávateľom. To umožňuje spresnenie zadania v akejkoľvek fáze projektu. Vďaka tomu sa znižuje riziko toho že projekt dopadne neúspešne.
- Používanie komponentovej architektúry - je založené na použití komponentov – zložiek architektúry, ktoré sú nezávislé, vymeniteľné a teda pomáhajú zvládať zložitosť systému a umožňujú znovupoužiteľnosť komplexnejších častí systému. Metodika RUP poskytuje metodickú, systematickú cestu návrhu, vývoja a overenie správnosti architektúry. Ďalej ponúka šablóny, architektonické štýly a pravidlá

designu. Nakoniec vďaka komponentom môže byť vývoj efektívnejší a znovupožitie komponentov zvyšuje úsporu zdrojov.

- Vizuálne modelovanie – systém sa zjednoduší do menších modelov, pričom abstrahuje od nedôležitých vlastností daného modelu. Potom sú takéto modely viac zrozumiteľné, a pri rozsiahlych projektoch nie je takmer možné pochopiť systém v celom jeho celku. RUP využíva k modelovaniu štandardný modelovací jazyk Unified Modeling Language (UML), čo prináša jednoduchšiu štandardizovanú komunikáciu v rámci týmu, ale i pri komunikácii so zákazníkom a zvyšuje konzistenciu projektu.
- Priebežná kontrola kvality – ak sa problém nájde a odstráni už v úvodných fázach návrhu (kde nám treba len previesť zmenu v niekoľkých diagramoch a špecifikáciách), je to nesmierne lacnejšie ako keby sa tento problém odstránil až v neskorších fázach vývoja projektu (kde už by bolo potrebné prepísať hotové časti programu). Preto RUP presadzuje kontrolu kvality a testovanie od počiatku projektu.
- Správa zmien – pri každom vývoji môžu nastať zmeny, pokiaľ zmeny neočakávame, tak nám narušia a rozvrátia celý vývojový proces. Ale ak sa na chyby pripravíme a integrujeme ich k vývoju, predídeme problémom.

Týmito praktikami sa riadi RUP vo svojich jednotlivých oblastiach [1,9].

5.2 Elementy RUP

RUP definuje štyri základné a najdôležitejšie elementy modelovania , na nich stavia celé modelovanie a prenesie cez ne aj celý vývojový proces.

Pracovníci (workers) a role (roles) – odpovedajú na otázku KTO? Pracovníka, je vhodné vidieť ako rolu, predstavuje viacero ľudí, ich zodpovednosti a chovanie. Zodpovednosti sú väčšinou vo vzťahu k artefaktom, ktoré sú kontrolované, vytvárané a modifikované pracovníkom. Chovanie zasa popisujeme pomocou činností.

Činnosti (activities) – odpovedajú na otázku AKO? Je vykonávaná jednotlivcom alebo skupinou v určitej role. Činnosť ovplyvňuje niekoľko artefaktov, má definované niekoľko vstupných a výstupných artefaktov. Ich účel je jasne definovaný, vyjadrovaný, ako vytvorenie či modifikácia artefaktu.

Artefakty (artifacts) – odpovedajú na otázku ČO? Je to časť informácie vyprodukovaná alebo modifikovaná v rámci procesu. Sú to hmatateľné výsledky projektu. Bývajú vo forme dokladov, zoznamov, požiadaviek, katalógov, modelov, diagramov, ktoré sa dodávajú ako vstup alebo výstup aktivity.

Pracovné procesy (workflows) – odpovedá na otázku KEDY? Je to postupnosť činností vedúcich k vytvoreniu požadovaného. Môže byť modelovaný v UML, ako sekvenčný diagram alebo ako diagram činností. Skladá sa z niekoľkých činností. RUP definuje deväť kľúčových procesov (Core Workflows) [1,10].

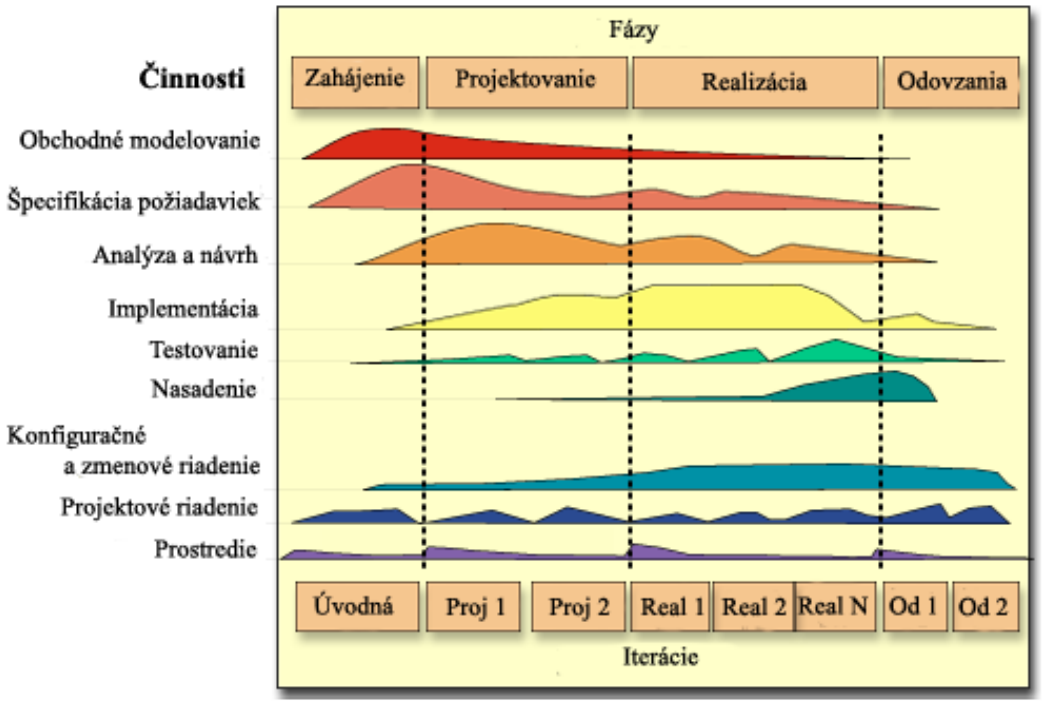
5.3 Postupnosť fáz

Vývoj prebieha v iteráciách, každá iterácia sa delí na 4 fázy.

Fázy iterácií :

- incepčná (inception),
- elaboračná (elaboration),
- konštrukčná (construction) ,
- prechodová (transition).

Vo fázach sa uskutočňujú iterácie. Počet iterácií je rôzny a záleží od rozsahu projektu a potrebách tímu [1,9].



Obr. 5 – Jeden vývojový cyklus v metodike RUP [20].

Fáza zahájenia (inception) – cieľom je dosiahnutie zhody cieľov projektu a životného cyklu. Určuje sa rozsah projektu, jeho hranice, prípady použitia, príprava a predvedenie architektúry, vyčíslenie nákladov a určenie rizík. Táto fáza má väčšinou len jednu iteráciu. Nevzniká žiadny softvér a uskutočňuje sa len pár plánovacích procesov.

Fáza projektovania (elaboration) – plánovanie činností, špecifikácia vlastností, navrhnutie architektúry, čiastočná analýza rizík, kontrola či architektúra, požiadavky a plány sú dostatočne stabilné a je tu vytvorený evolučný prototyp. Často krát má dve iterácie. Ak je však nutné do projektu zapracovať nových pracovníkov, prípadne predpokladáme väčší vznik rizík, môže mať tri až štyri iterácie.

Fáza realizácie (Construction) – zahŕňa rozvíjanie architektúry a plánov, objasňovanie požiadaviek. Skompletizovanie analýzy a designu. V tejto fáze je možné dodávať prototypy, alfaverzie a betaverzie. Prebieha testovanie a vyhodnotenie kvality produktu a plnenie špecifikácií. Ďalej sa rozhoduje či je softvér dostatočne otestovaný a pripravený na odovzdanie zákazníkovi. Plánujú sa poväčšine aspoň dve iterácie. V prípade, že potrebujeme dôkladnejšiu integráciu a testovanie, môžeme očakávať tri iterácie. Ak ide o rozsiahly projekt iterácii môže mať aj štyri iterácie.

Fáza odovzdania (Transition) – odovzdanie produktu zákazníkovi. Táto fáza obsahuje aj školenie, podporu a údržbu pokiaľ nie je používateľ spokojný. Ďalej sa v tejto fáze počíta aj s upravovaním a vylepšením produktu. Ak máme len betaverziu a plnú verziu, iterácie sú dve, no v prípade nájdenia ďalších chýb, je nutné pridať ďalšiu iteráciu [1].

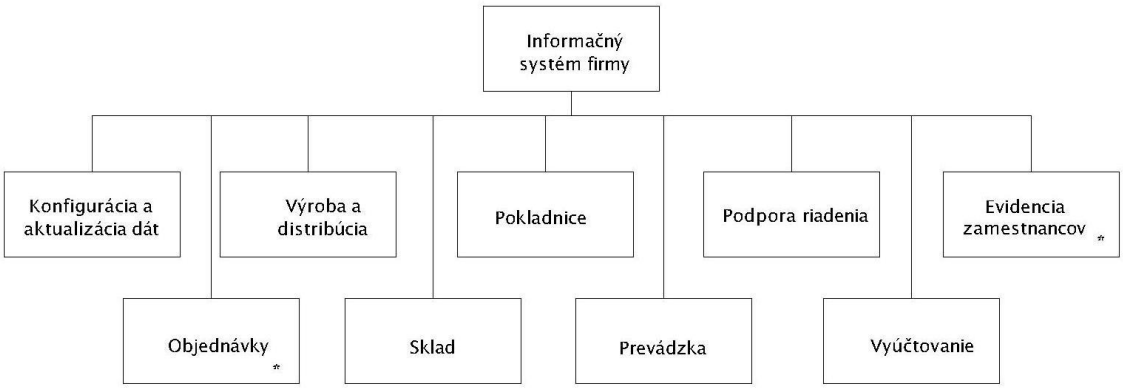
6 Procesné modelovanie

Od procesného modelu sa odvádzajú základné požiadavky na informačný systém. Procesný model najprv opisuje hlavné procesy zákazníka, objasňuje nás s tým ako fungujú v súčasnosti, aby sa následne identifikovali tie časti procesov, ktoré majú byť podporované novým informačným systémom. Pritom sa od procesov prechádza na jednotlivé činnosti, ktoré majú byť vykonávané s podporou informačného systému. Z nich sa odvádzajú funkčné požiadavky ako aj požiadavky na používateľské rozhranie zodpovedajúce zručnostiam pracovníkov vykonávajúcich činnosť [11].

Procesné modelovanie sa používa ako úvodný krok pri zahájení analytických prác. Je počiatočnou fázou väčšiny softvérových projektov. Procesným modelovaním je možné zabrániť, že zákazník nepomyslí na rôzne aspekty informačného systému a unikne nám veľa požiadaviek a súvislostí, ktoré môžu návrh informačného systému značne a významne ovplyvniť. Preto je vhodne využiť diagramy firemných procesov už pri tvorbe úvodnej štúdie [12].

6.1 Diagramy hierarchie procesov

Diagram hierarchie procesov (Process Hierarchy Diagram) rieši procesný rozpad systému. Pomáha ujasniť rozsah vyvíjaného systému, jeho modularitu a súvislosti vzájomných firemných procesov na najvyššej úrovni. Pričom firemný proces je sekvencia činností vytvárajúci produkt, ktorý organizácia potrebuje pre firemné ciele. Na obrázku č.6 je príklad takéhoto diagramu pre hierarchiu procesov informačného systému výrobnej firmy. Diagram obsahuje rozpad prvej úrovne [12].



Orb. 6 – Diagram hierarchie procesov [12].

6.2 Diagram procesných vlákien

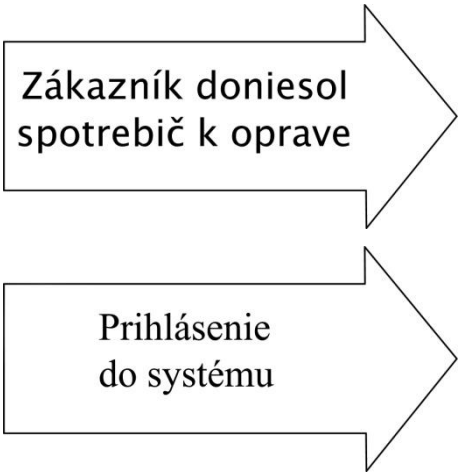
Diagram procesných vlákien (Process Thread Diagram - PTD) sa využíva na celkový pohľad na podnikový informačný systém z hľadiska dynamiky. Veľká výhoda takéhoto diagramu je jeho zrozumiteľnosť ako pre zadávateľa, tak i pre zákazníka. Touto metódou sa modelujú inicializačné firemné procesy firemných udalostí a taktiež výsledky činnosti takýchto procesov.

Ďalšie plus je možnosť mapovania vzájomne prepojených firemných procesov. Pričom firemný proces je definovaný ako aktivita, ktorá sa vykonáva ako odozva na firemnú udalosť poväčšine na jednom mieste a v jednom čase.

Symbol používaný pre firemné procesy v diagramoch procesných vlákien je uvedený na obr. č. 7.

Obr. 7 – Symbol firemného procesu na diagramoch [12].

Firemná udalosť (Business Event) je reprezentačnou udalosťou pre problémové oblasti firmy, ktoré sú štartérom pre rôzne aktivity. Takáto firemná udalosť môže byť externá, alebo interná. (značenie: externá – prečiarknutá šípka; interná – šípka bez prečiarknutia). Úlohou firemnej udalosti je spustenie príslušného firemného procesu. Symbol pre firemnú udalosť, ktorá spustí pre ňu zodpovedný firemný proces sa nachádza na obr. č. 8.

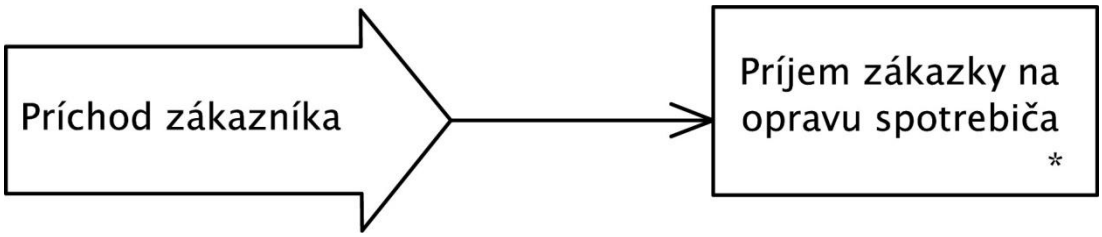


Obr. 8 - Symbol pre udalosť [12].

Procesný reťazec je ukončením procesných výsledkov. Firemný výsledok je znázornený na obr. č. 9.

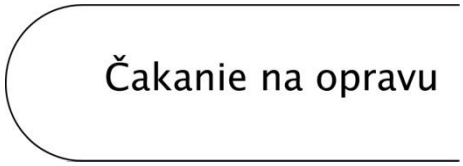
Obr. 9 – Symbol pre firemný výsledok [12].

Závislosť procesov nefiremných udalostí je graficky znázorňovaná ako čiara spájajúca tieto dva symboly. Pokiaľ nastane určitá udalosť, musí byť pre ňu vykonaný proces. Táto závislosť je reprezentovaná šípkou medzi udalosťou a procesom. Príklad zobrazenia je na obr.č.10.



Obr. 10 – Firemná udalosť inicializuje firemný proces [12].

Ďalším symbolom je prerušenie procesu, ktoré indikuje, že procesy boli pozastavené a ďalej môžu znova prebiehať až keď nastane nová udalosť. Kým táto udalosť nevznikne sú procesy pozastavené. Symbol je zobrazený na obr.č.11.



Obr. 11 - Symbol pre prerušenie [12].

Súbeh procesov (nadproces) je združenie procesov prebiehajúcich v poradí , ktoré však nie je pre nás zaujímavé. O takomto nadproces-e hovoríme aj ako združení paralelných procesov. Môže obsahovať aj podmienku pre opakované vykonávanie súbehu procesov.

Závislosť podobná súbehu procesov je nazývaná aj ako exkluzívna závislosť. Používa sa pre znázornenie toho, že len jeden proces z viacerých je spustený [12].

7 Dátové modelovanie

Dátové modelovanie je metóda ktorá slúži na definovanie a analyzovanie dátových požiadaviek, ktoré sú potrebné pre procesy organizácie. Dátové požiadavky sú zaznamenávané ako konceptuálny dátový model s definíciami dát. Ďalej definuje vzťahy medzi dátovými prvkami a štruktúrami. Táto metóda sa odporúča pre všetky projekty, ktoré vyžadujú štandardné spôsoby ako definovať a analyzovať dátové zdroje v rámci organizácie [13,14].

Používané metodiky rozdeľujú pohľad na dáta, na logický a fyzický dátový model [12].

7.1 Logický dátový model

Logický dátový model popisuje uložené údaje a vzťahy medzi nimi. V logickom modeli sa vyskytujú pojmy: entita , atribút a väzba.

Entita: Jedná sa o dátový objekt o ktorom zhromažďujeme informácie. Entitu popisujeme podstatným menom. Je to určité zoskupenie dát, ktoré k sebe logicky prináležia a ktoré skúmame v rámci zadanej analýzy.

(Príklady entít : pacient (v nemocnici) , kniha (v knižnici) , lístok (doprava / divadlo))

Atribút: Atribút je vlastnosť alebo dá sa povedať aj položka danej entity. Jedna entita môže mať viacero atribútov.

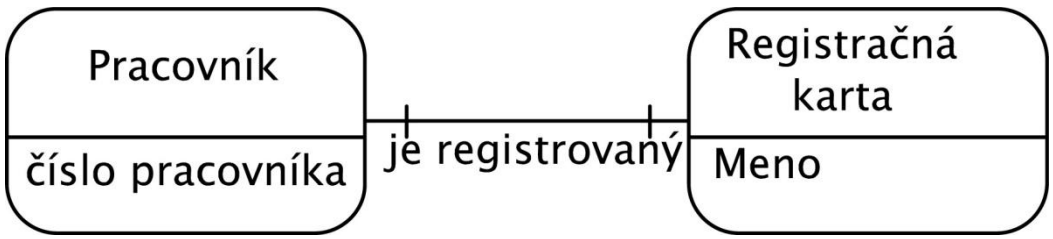
(napr. atribúty entity pacient sú meno , priezvisko , rod. Č. ,adresa .....)

(napr. atribúty entity kniha sú názov , autor , žáner , rok vydania .....)

(napr. atribúty entity lístok sú cena , sála , dátum predstavenia ,názov predstavenia .....)

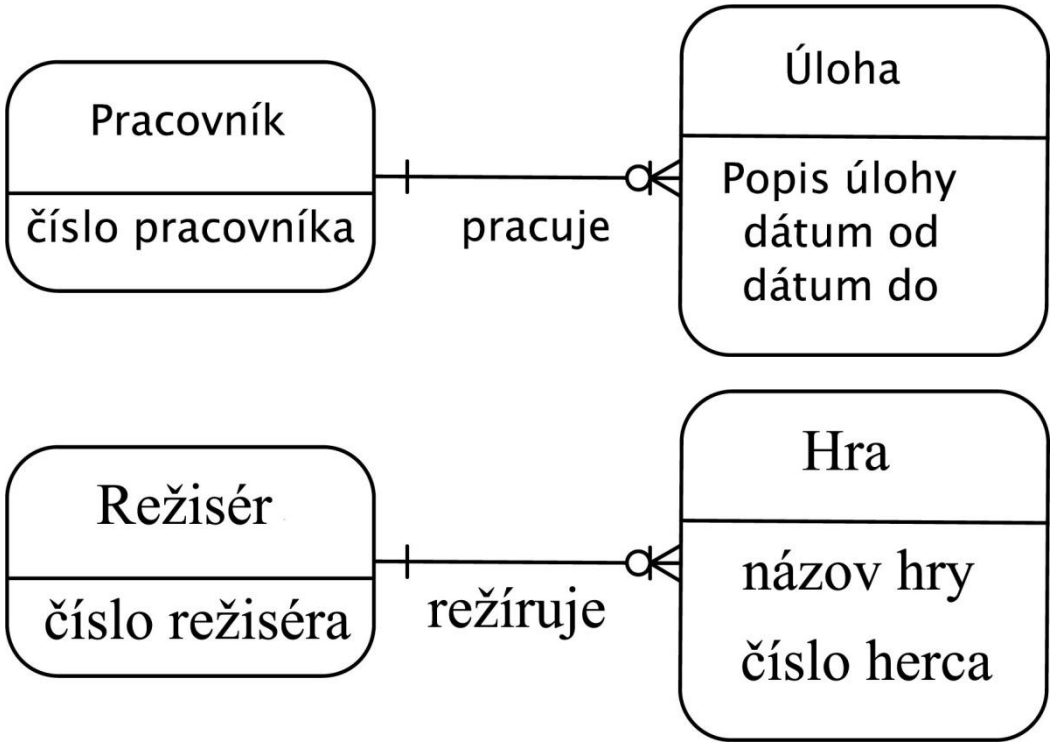
Väzba (vzťah): Väzba určuje vzťah medzi entitami, poznáme tri typy.

Väzba typu 1:1: Pri takomto type väzby vystupuje na každej strane len jeden objekt entity. Tento druh väzby sa nevyskytuje veľmi často pretože často vedie k zlúčeniu entít (napr. jeden pracovník ma jednu registračnú kartu).



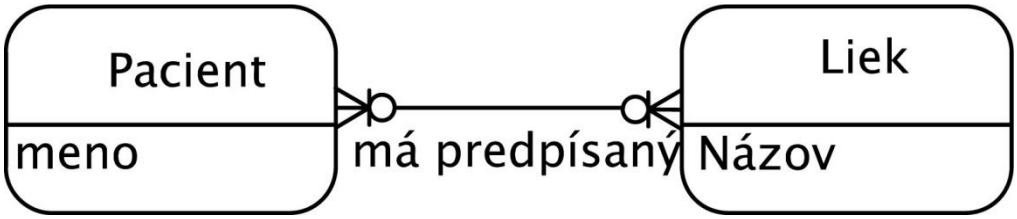
Obr. 12 – Väzba medzi entitami typu 1:1 [12].

Väzba typu 1:N: Jedná sa o najčastejší druh väzby. Pričom na jednej strane sa nachádza len jeden objekt danej entity a na strane druhej ich môže byť viac. Napríklad také trvalé bydlisko má každý človek uvedené len jedno, ale na danom bydlisku žije viac obyvateľov (napr. jeden pracovník má priradených viacero úloh).



Obr. 13 - Väzba medzi entitami typu 1:N [12].

Väzba typu M:N: V tomto vzťahu vystupujú na oboch stranách väzby viac objektov danej entity. Používa sa len v začiatočných fázach dátovej analýzy. Neskôr musí byť prevedená na väzby typu 1:N (dekompozícia väzby M:N) (napr. pacient môže mať predpísaných viac liekov a liek môže byť predpísaný viacerým pacientom).



Obr. 14 - Väzba medzi entitami typu M:N [12].

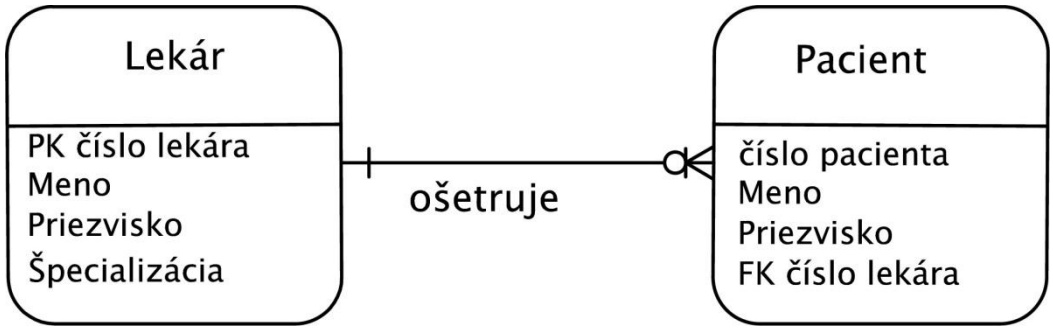
Kľúče : Každá entita má svoje atribúty, pričom jeden (alebo viac) ma významnejšiu úlohu ako ostatné. Takýto atribút sa nazýva kľúč. Kľúče delíme na tri typy.

Typy kľúčov :

Primárny kľúč (PK = Primary key): Takýto kľúč musí byť jednoznačný (jedinečný), to znamená, že výskyt kľúča sa musí od ďalších výskytov takéhoto kľúča líšiť, tak aby jednoznačne identifikoval každý výskyt entity. Ako príklad PK, by mohlo poslúžiť rodné číslo v entite OSOBA. Pretože mená, či priezviská môžu mať aj dvaja iný ľudia totožné, pričom rodné číslo by malo byť jedinečné pre každú osobu.

Alternatívny kľúč (alternative key): Sú také kľúče, ktoré tiež jednoznačne identifikujú danú entitu, no však neboli ako primárny kľúč vybrané. Ale sú ďalšou možnosťou (alternatívou) pre primárny kľúč.

Cudzí kľúč (CK / FK = foreign key): Je to taký kľúč, ktorý je ako atribút v jednej entite A, ale vyskytuje sa v inej entite B ako kľúč primárny. (napr. entita pacient obsahuje ako atribút číslo lekára pričom číslo lekára je PK v entite lekár. Teda číslo lekára je CK/FK pre entitu pacient)



Obr. 15 – Primárne a cudzie kľúče [12].

7.2 Fyzický dátový model

Fyzický datový model je realizáciou logického modelu v prostredí informačného systému. Pre fyzické modelovanie sú dôležité nasledovné pojmy:

Databáza – organizovaná štruktúra dát usporiadaná do tabuliek.

Tabuľka – dátová štruktúra organizovaná do matice riadkov a stĺpcov, obsahujúca dáta.

Riadok – obsahuje informácie o jednom výskyte entity v databáze.

Stĺpec – je vlastne atribút z logického dátového modelu.

Normalizácia dátového modelu je založená na matematických pravidlách. Oddeľuje logický a fyzický model . Postup pre normalizáciu sa skladá z nasledujúcich krokov:

1. prevod dát do nenormalizovanej tabuľky – vyberie sa PK pre každú nenormalizovanú tabuľku.
2. prevod do 1. normálnej formy – v tejto forme nesmú tabuľky obsahovať opakujúce sa skupiny, takéto skupiny sa potom prevedú do vlastnej tabuľky.
3. prevod do 2. normálnej formy – tabuľka nesmie obsahovať stĺpec, ktorý závisí len na časti zloženého kľúča. V jednej relácii sa nesnažíme zachytiť viac entít.
4. prevod do 3. normálnej formy – je nutné odstrániť závislosti medzi nekľúčovými stĺpcami a vzájomné závislosti v rámci kľúča.
5. konsolidácia tabuliek v 3. normálnej forme – počas normalizácie môžu vzniknúť tabuľky s rovnakými kľúčmi. Takéto tabuľky prevedieme znova do 3. normálnej formy [12].

8 UML

Do roku 1994 existovalo viac metód pre vizuálne modelovanie. Každá metodika priniesla niečo nového i novú kvalitu. Väčšinu trhu ovládli metódy BooCH a OMT (object modeling technique). Prvý pokus o zlúčenie bola metodika Fusion. V roku 1996 združenie OMG (Object Management Group) navrhlo špecifikáciu RFP (Request For Proposal) pre objektovo orientovaný jazyk pre vizuálne modelovanie, v ktorom jazyk UML navrhla ako štandard. Odvtedy bolo UML prijaté verejnosťou.

UML (Unified Modeling Language, unifikovaný modelovací jazyk) je univerzálny jazyk pre modelovanie systémov. Bol navrhnutý tak, aby spojil postupy modelovacích techník a softvérového inžinierstva. Diagramy UML sú zrozumiteľné ľuďom a je ich aj možné interpretovať pomocou nástrojov CASE [16].

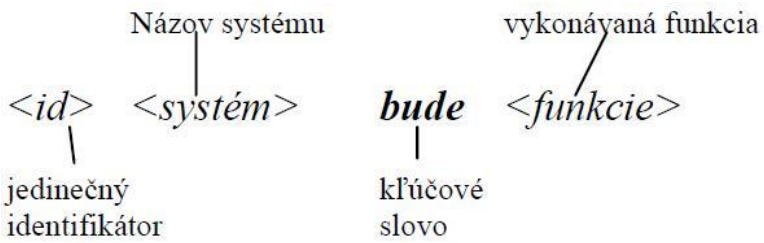
UML slúži na vizuálne modelovanie vytváraného softvéru. Svoje využitie nájde predovšetkým počas fáz analýzy a návrhu softvéru, pričom poskytuje prehľadný a jasný zápis pomocou diagramov. V súčasnosti tento jazyk predstavuje štandard pri tvorbe softvéru [19].

8.1 Požiadavky

Požiadavky sú špecifikácia toho čo budeme implementovať. Hovoria nám čo by sme mali vytvoriť, no nie ako by sme to mali urobiť. Poznáme dva typy požiadaviek [16]:

- Funkčné požiadavky – ktoré určujú, aké správanie bude systém ponúkať.
- Nefunkčné požiadavky – ktoré špecifikujú vlastnosti alebo obmedzujú podmienky daného systému.

Pre zápis požiadaviek sa používa jednoduchý predpis (obr.č.17).



Obr. 16 - Predpis pre zápis požiadaviek [17].

Funkčná požiadavka je formulácia toho čo by mal systém robiť - popisuje požadovanú funkciu systému. Ak napríklad zhromažďujeme požiadavky pre bankomat, môžu sa medzi funkčnými požiadavkami objaviť i takéto:

1. Bankomat bude overovať platnosť vloženej karty.
2. Bankomat bude overovať platnosť PIN kódu zadaného zákazníkom.

Nefunkčná požiadavka je obmedzujúca podmienka uvalená na daný systém. U bankomatu by sa mohli v zozname nefunkčných požiadaviek objaviť nasledujúce [16]:

1. Riadiaci systém bankomatu bude naprogramovaný v jazyku C++.
2. Riadiaci systém bankomatu bude overovať platnosť PIN kódu maximálne do troch sekúnd.

8.2 Model prípadov použitia (Use case model)

Presne zachytávajú funkčnosť, ktorá bude budúcim informačným systémom pokrytá a vymedzujú tak jednoznačne rozsah prác. Vyvinutý systém nebude obsahovať nič iné, než popisujú prípady použitia. Každý prípad použitia popisuje jeden zo spôsobov použitia systému, popisuje teda jednu jeho požadovanú funkčnosť [18].

Výstupom uvedených aktivít je model prípadov použitia. Tento model obsahuje štyri komponenty:

- Účastníci (actors) – sú to role, pridelené osobám alebo predmetom používajúce daný systém.
- Prípady použitia (use cases) – činnosti, ktoré môžu účastníci so systémom vykonávať.
- Relácie (relationships) – zmysluplné vzťahy medzi účastníkmi a prípadmi použitia.
- Hranice systému (system boundary) – ohraničenie zobrazené okolo prípadov použitia, tiež je vyznačením územia alebo hraníc modelovaného systému.

Modelovanie prípadov použitia je doplňujúcim spôsobom špecifikácie a dokumentácie požiadaviek. Modelovanie prípadov použitia prebieha v nasledujúcich krokoch [17]:

- Nájdenie hraníc systému – určíme, čo je súčasťou systému, a čo už nie je. Hranice systému špecifikuje účastník, ktorý systém používa.
- Vyhľadanie účastníkov (actors) – aktér je ten, kto so systémom priamo komunikuje, môže sa jednať o študenta v knižnici, zákazníka nakupujúceho cez elektronický obchod alebo o senzor v elektrárni. Aktér je rola, nejedná sa o človeka

alebo vec. Jeden človek môže hrať viacero rôznych rolí, môže byť aj študentom a aj zároveň zákazníkom v e-shope.

- Nájdenie prípadov použitia – prípad použitia je niečo, čo účastník očakáva od systému. Je vždy iniciovaný účastníkom. Prípad použitia popisuje nejakú akciu.
- Špecifikácia prípadov použitia – podrobnejší popis prípadov použitia.
- Tvorba scenárov.

8.2.1 Tvorba scenárov

Pre špecifikáciu prípadov použitia neexistuje žiadny základný štandard UML. Obvykle sa však používa šablóna zobrazená na obr.č.17 [16].

názov prípadu použitia -	**Prípad použitia : PlatitDanZPridanejHodnoty**
jedinečný identifikátor -	**ID : UC1**
účastníci prípadu použitia -	**Účastníci:** Čas finančný úrad
stav systému pred spustením prípadu použitia -	**Vstupné podmienky:** 1. Je koniec fiškálneho štvrťroku?
skutočné kroky prípadu použitia -	**Tok udalostí:** 1. Prípad použitia začína na konci fiškálneho štvrťroku 2. Systém určuje výšku dane z pridanej hodnoty, ktorú je treba odviesť štátu 3. Systém odosiela elektronickú platbu finančnému úradu
stav systému po ukončení prípadu použitia -	**Výstupné podmienky:** 1. Finančný úrad príjme daň z pridanej hodnoty správnej hodnote

Obr. 17 – Slovný scenár [16].

Každý prípad použitia má svoj názov a špecifikáciu. Špecifikácia sa skladá z nasledujúcich častí [16]:

- Vstupné podmienky – kritériá, ktoré musia byť splnené ešte predtým, ako je možné spustiť prípad použitia. Sú to obmedzenia stavu systému.
- Tok udalostí – jednotlivé kroky v prípade použitia.
- Výstupné podmienky – kritériá, ktoré musia byť splnené na konci prípadu použitia.

8.2.2 Relácie

Medzi účastníkmi a prípadmi použitia môžu vzniknúť nasledujúce vzťahy:

Generalizácia (zovšeobecnenie) - o zovšeobecnení účastníka môžeme uvažovať, keď majú dvaja alebo viacerí účastníci mnoho spoločného. Jedná sa predovšetkým o spôsob interakcie so systémom, t.j. ak spúšťajú účastníci rovnaké prípady použitia. Pre zovšeobecnenie účastníkov platí jedna dôležitá podmienka: Potomka môžeme dosadiť všade tam, kde možno očakávať prítomnosť (kde sa môže vyskytovať) jeho predka. Rodičovský účastník môže byť buď abstraktná alebo, konkrétna rola. Potomkovia dedia od svojich predkov nielen role, ale i relácie a z toho dôvodu musí platiť uvedená podmienka.

Include - umožňuje zahrnúť chovanie dodávateľského prípadu použitia do toku klientskeho prípadu. Klientsky prípad použitia nie je bez zahrnutia dodávateľských prípadov úplný. Dodávateľské prípady tvoria nevyhnutnú súčasť klientskeho prípadu. V klientskom prípade použitia musíme určiť presný bod, kedy dodávateľský prípad zahrnúť.

Extend - poskytuje spôsob rozšírenia chovania pôvodného prípadu použitia o chovanie nové. Bázový prípad použitia poskytuje určitú množinu bodov, ktorým hovoríme *body rozšírenia* (extension points). K ním možno pripojiť rozšírenie v podobe nového chovania. Bázový prípad použitia je úplný i bez rozširujúcich prípadov. Pre ne má pripravené body rozšírenia a vôbec o rozširujúcich prípadoch nemusí vedieť. Tok udalostí bázového prípadu použitia nevie, v akom mieste bude rozšírený. Body rozšírenia totiž existujú vo vrstve nad hlavným tokom udalostí [17].

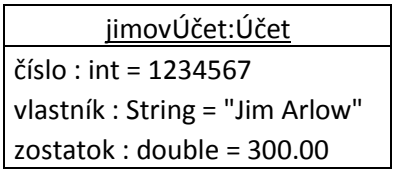
8.3 Objekty

James Rumbaugh v knihe „The Unified Modeling Language Reference Manual“ (Referenčná príručka jazyka UML) definuje objekt ako „diskrétnu entitu s jasne definovaným rozhraním, ktoré zapuzdruje stav a chovanie – ako inštanciu triedy“.

Objekt kombinuje dáta a funkcie do jednej súdržnej jednotky. Ukrýva svoje dáta za vrstvami funkcií. Každý objekt sa dá identifikovať jedinečným spôsobom. Hodnoty atribútov uchovávajú dáta objektu.

8.3.1 Notácia objektov v jazyku UML

Symbolom objektu v jazyku UML je obdĺžnik s dvoma časťami (obr.č.18). Vrchná časť obsahuje identifikátor objektu, ktorý je vždy podčiarknutý.



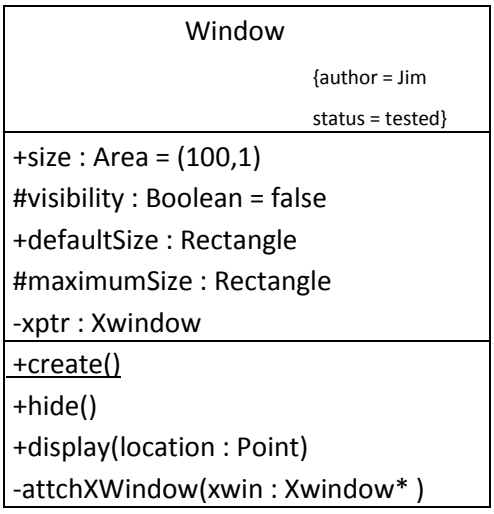
Obr. 18 – Objekt v jazyku UML [16].

8.4 Triedy

V knihe „The Unified Modeling Language Reference Manual“ (Referenčná príručka jazyka UML) je trieda definovaná ako „deskriptor množiny objektov, ktoré zdieľajú rovnaké atribúty, operácie, metódy, relácie a správanie“. Znamená to, že trieda je deskriptorom množiny objektov rovnakými charakteristickými vlastnosťami. Trieda popisuje charakteristické rysy určitej množiny objektov. Každý objekt je inštanciou presne jednej triedy.

8.4.1 Notácia tried v jazyku UML

Grafická syntax jazyka UML je pre triedy veľmi bohatá. Obsahuje nepovinné ozdoby (adornments). Jedinou povinnou časťou grafickej syntaxe je oddiel symbolov s názvom triedy [16].



Obr. 19 – Trieda v jazyku UML [16].

9 CASE

Skratka CASE (z anglického Computer Aided Software Engineering alebo Computer Aided Systems Engineering) čo v preklade znamená počítačom podporované softvérové inžinierstvo alebo tiež vývoj softvéru s využitím počítačovej podpory.

CASE nástroje primárne umožňujú :

- Generovanie zdrojového kódu z modelu.
- Spätné vytvorenie modelu podľa existujúceho zdrojového kódu.
- Synchronizáciu modelu a zdrojového kódu.
- Vytvorenie dokumentácie podľa modelu.

CASE nástroje boli postavené tak, aby podporovali tímovú spoluprácu pri vývoji systému. Mimo iného zabezpečujú správu vývoja, sledujú konzistenciu modelu systému, automatizujú niektoré procesy pri tvorbe softvéru, niektoré umožňujú riadenie celého životného cyklu vývoja softvéru.

História prvých CASE nástrojov siaha až do 70. rokov, spoločne s rozvojom štruktúrnej a objektovej analýzy. Medzi prvé nástroje patrili tzv. Lower case nástroje určené hlavne pre fázu implementácie, ktoré boli schopné generovania kódu. Pôvodná myšlienka CASE nástrojov bola taká, že programátori nebudú potrební a CASE nastroj vygeneruje samotný kód, ktorý už nebude treba upravovať. Táto myšlienka však bola nesprávna a spôsobovala problémy nekonzistentnosti návrhu a kódu pri zmene jednej z týchto veličín. Presadzovanie CASE nástrojov sa začína v 80. rokoch spolu s grafickým rozhraním (GUI). V tomto období sa formujú jednotlivé CASE nástroje, hlavným dôvodom bola častá a často krát zložitá aktualizácia diagramu/modelu. V súčasnej dobe existuje nespočetné množstvo CASE nástrojov, ktoré plnia rôzne funkcie, nepomáhajú už iba programátorom, ale pomáhajú aj napr. pri riadení firmy, kde CASE nástroje analyzujú procesy firmy a za pomoci nástrojov sa tieto procesy modelujú a upravujú tak akoby mali správne fungovať. Väčšina CASE nástrojov súčasnosti, predstavuje veľké aplikácie, ktoré môžeme použiť vo všetkých fázach vývoja softvéru. U väčšiny z nich je jadrom jazyk UML [15].

Komponenty CASE nástrojov

- GUI – grafické prostredie pre vytváranie modelov.
- Centrálna databáza objektov- zaručuje možnosť použiť daný objekt v ľubovoľnom ďalšom kroku modelovania.
- Prostriedky verifikácie a konzistentnosti dát.

- Textový editor pre popis objektov.
- Generátor zdrojového kódu.
- Export / import.

Rozdelenie podľa životného cyklu projektu

- Pre CASE (podporujú činnosti predchádzajúce vývoju IS).
- Upper CASE (podporuje tvorbu fázy analýzy).
- Middle CASE (podporuje tvorbu detailného návrhu IS).
- Lower CASE (podporuje fázu implementácie).
- Post CASE (podporuje fázu uvedenia IS do prevádzky a jeho údržbu).

Rozdelenie podľa fázy projektu

- Upper CASE (nástroje podporujúce fázu analýzy a návrhu).
- Lower CASE (nástroje podporujúce fázu implementácie a testovania).

Rozdelenie podľa interaktivity

- CASE astride, ktoré sú interaktívne zo svojej podstaty (napr. nástroje podporujúce metódu návrhu).
- CASE nástroje, ktoré nie sú interaktívne (tzv. vývojové nástroje, napr. prekladače).

Rozdelenie podľa fázy projektu vývoja softvéru, v ktorej sú využívané

- front-end CASE nástroje (využívané v začiatočných fázach projektu – napr. nástroje na podporu návrhu).
- back-end CASE nástroje (využívané v začiatočných fázach projektu – napr. kompilery a nástroje podporujúce testovanie).

Rozdelenie podľa toho, či sú využívané počas celého životného cyklu softwaru

- vertikálne CASE nástroje (nástroje podporujúce len určitý krok životného cyklu softvéru či určitej oblasti – napr. zisťovanie používateľských požiadaviek alebo kódovanie).
- horizontálne CASE nástroje (nástroje podporujúce niekoľko krokov životného cyklu softvéru či viac oblastí – napr. nástroje pre tvorbu dokumentácie či riadenie konfigurácie).

Rozdelenie podľa stupne integrácie

- CASE tools sú nástroje zabezpečujúce automatizovanú podporu ľubovoľnej úlohy životného cyklu softvéru.
- CASE toolkits je súbor integrovaných softvérových nástrojov, ktorý poskytuje čiastočnú či komplexnú podporu len v rámci jednej fázy životného cyklu softvéru.

- CASE workbenches je množina integrovaných CASE tools alebo CASE toolkits, ktorá poskytuje čiastočnú či komplexnú podporu v minimálne dvoch fázach životného cyklu softvéru.
- I – CASE predstavuje najvyšší stupeň integrácie; prepojenie niekoľko CASE tools, CASE toolkits a CASE workbenches.

Prínosy CASE nástrojov: Vyššia produktivita práce, nižšia chybovosť, lepšia a rýchlejšia údržba a ďalší vývoj výsledného produktu, kvalitnejšia dokumentácia, umožňuje spoluprácu väčšieho počtu ľudí na vývoji produktu.

Príklady CASE nástrojov: Powerdesigner (Sybase), Oracle Designer (Oracle), Case Studio, Rational Rose, MS Visio [15].

Pre náš návrh sme si zvolili CASE nástroj DIA v0.96.1.

10 Návrh IS Divadlo

Súčasná technická doba v ktorej žijeme, zameraná na internetovú komunikáciu, núti divadlá, aby sa transformovali a ponúkli zákazníkom nové možnosti ako rýchlo, jednoducho a v pohodlí domova rezervovať lístok na obľúbené divadelné predstavenie a to bez nutnosti státia v nekonečných radoch na lístok pred divadlom. Ďalšia problémová oblasť sa tyká samotnej prípravy predstavenia. Vkus samotného diváka sa často rýchlo mení a hra, ktorá bola zaujímavá minulé roky nemusí byť zaujímavá aj dnes, čo môže spôsobiť divadlu veľké finančné straty. Preto je nevyhnutne rýchlo identifikovať zmeny a prispôsobiť sa vkusu diváka. Príprava divadelnej hry predstavuje zdĺhavý a náročný proces v ktorom dramaturg najprv vyberie režiséra, aby zostavil tým hercov pre konkrétnu divadelnú hru. Počas nasledujúcich niekoľkých týždňov sa organizuje výber hercov pre konkrétne roly. Pre každú rolu sa vyberie väčší počet hercov, ktorí sa počas týždňových alebo dvojtýždňových cyklov hrania hry striedajú. Nacvičovanie prebieha v dvoch fázach. Prvá fáza prebieha v priestoroch mimo divadelnej sály, kedy sa herci oboznamujú so svojimi rolami, tu sa nacvičia samotné základy, aby sa mohlo prejsť na nacvičovanie v divadelnej sále. Tu sa začína interakcia medzi režisérom a ľuďmi, ktorí majú technicky zabezpečiť predstavenie. Kulisár musí na základe konzultácie nechať pripraviť kulisy, tie sa vyrábajú priamo na zákazku. Takisto kostymér vytvorí kostýmy pre konkrétnych hercov priamo na mieru, tu sa kladie veľký dôraz na dobovú autentickosť. Počas niekoľkých ďalších mesiacov až rokov sa predstavenia hrávajú v pravidelných intervaloch. V prípade poklesu záujmu sa doba medzi jednotlivými predstaveniami predlžuje, cena lístkov klesá, až sa napokon predstavenie vyradí z ponuky divadla.

10.1 Požiadavky

Funkčné požiadavky :

1. Systém bude vytvárať rozvrhy nácvikov.
2. Systém bude umožňovať priraďovanie hercov na nácviky.
3. Systém bude umožňovať priraďovanie hercov na predstavenia.
4. Systém bude poskytovať prehľad nácvikov a predstavení.
5. Systém bude vytvárať program divadla.
6. Systém bude umožňovať rezerváciu / objednanie lístka.
7. Systém bude evidovať priestory, predstavenia, zamestnancov, rekvizity a informácie o zákazníkoch.
8. Systém bude umožňovať priradenie režiséra k hre.
9. Systém bude poskytovať možnosť na preloženie hry.
10. Systém bude vytvárať zálohu údajov.

Nefunkčné požiadavky :

1. Na jedno miesto je možné objednať len jeden lístok.
2. Na zrušené predstavenia nebude možné lístok objednať/rezervovať.
3. V jednej sále môže byť len jedno predstavenie v danú dobu.
4. Herec môže hrať len v jednom predstavení/nácviku v danú dobu.
5. Údaje v systéme budú dodržiavať ochranu osobných údajov.
6. Interakcia so zákazníkom bude používať webové rozhranie.
7. Zamestnanec sa do systému prihlási čipovou kartou.
8. Objednanie/rezervovanie lístka prebehne do 3 sekúnd.
9. Objednanie/rezervovanie lístka musí byť aspoň 12 hodín pred predstavením.
10. Prihlásenie do systému prebehne do 3 sekúnd.

10.2 Aktéri

- Dramaturg – zaoberá sa zaraďovaním hier. Ak je hra nová, zaeviduje hru do systému, predtým zistí či je hru potrebne preložiť.

- Režisér – vyberá a eviduje hercov na predstavenia a nácviky, taktiež zabezpečuje kulisy a kostýmy ktoré sú pre dané predstavenie (nácvik) potrebné.

- Neprihlásený zákazník – si môže len prezerať hry a informácie o nich. Prípadne sa prihlásiť/registrovať.

- Prihlásený zákazník – objednáva/rezervuje si lístok na predstavenia podľa vlastného uváženia a rozhodnutia.

- Administrátor – ma plný prístup k celému informačnému systému.

- Čas – každý deň zálohuje údaje.

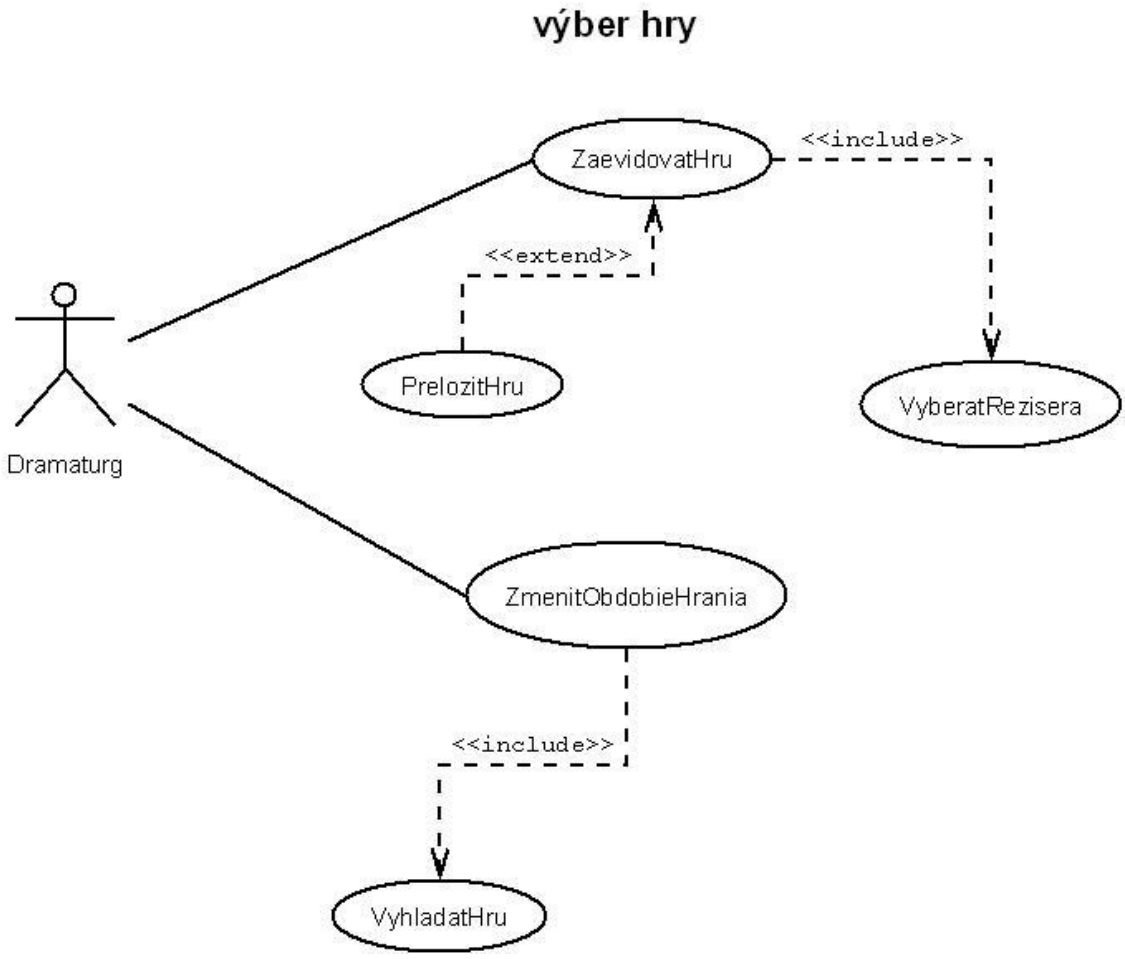
- Zamestnanec – všetci ostatní zamestnanci, ktorí sa podieľajú na behu divadla. Napríklad : herec , maskér , kulisár, atď..

- Objednávateľ – abstraktný aktér pre zjednodušenie diagramu prípadu použitia „Sprava sály”.

10. 3 Diagramy prípadov použitia (Use-case diagrams)

10.3.1 Výber hry

Dramaturg najprv zadá názov hry do systému, aby zistil či sa táto hra už niekedy v divadle hrala alebo je to úplne nová, v divadle ešte nehraná hra. V prvom prípade systém zobrazí informácie o hre (kedy a ako dlho sa hrala) a dramaturg iba zmení obdobie hrania. V prípade novej hry sa zistí, či treba dať hru preložiť.

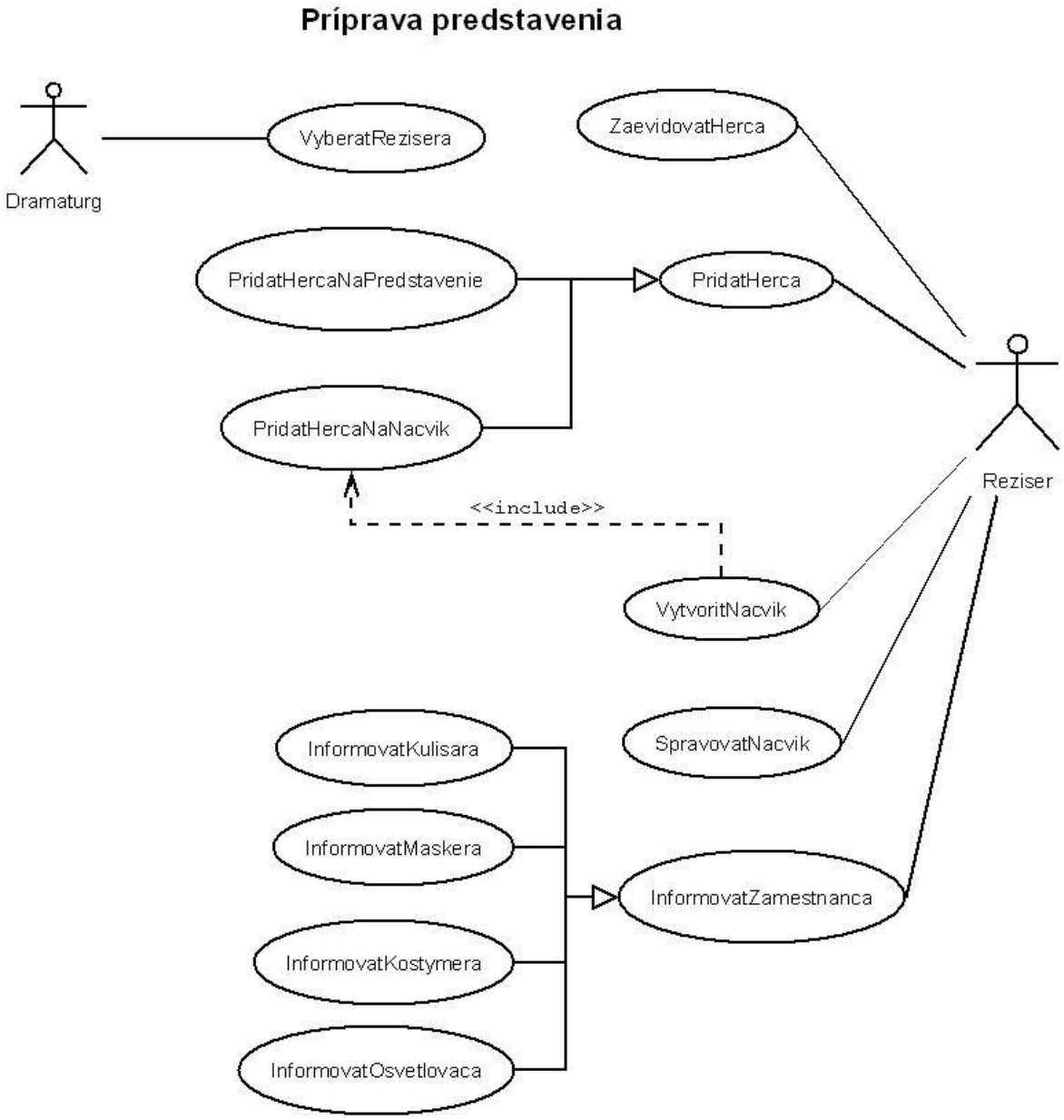


Obr. 20 – Výber hry.

10.3.2 Príprava predstavenia

Dramaturg najprv vyberie režiséra. Úlohou režiséra je v prvom rade vybrať hercov a zaevidovať ich do systému. Režisér vyberá hercov a zadáva do systému informácie o nich a ich rolách, ktoré neskôr systém uloží. Potom režisér vytvorí rozvrhy nácvikov. Zadá do systému miesto a čas nácvikov a do každého pomocou systému pridáva hercov zo zoznamu hercov podľa toho, ktoré scény sa budú nacvičovať.

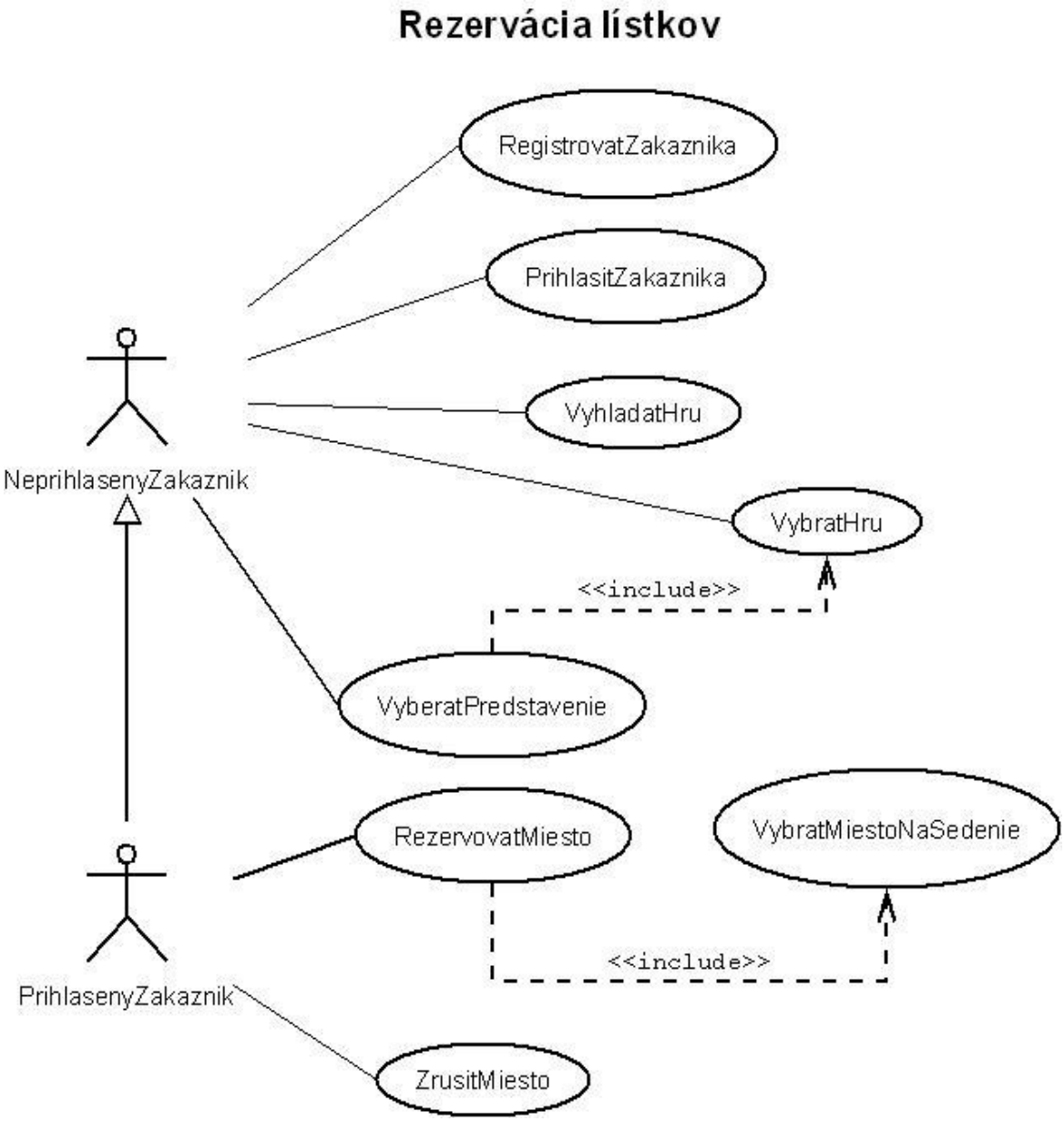
Režisér zapíše do systému taktiež informácie pre ďalších zamestnancov divadla, ktorí sú potrební pre pripravenie javiska. Určí, ktoré kulisy a kostýmy sú na daný nácvik potrebné.



Obr. 21 – Príprava predstavenia.

10.3.3 Rezervácia lístkov

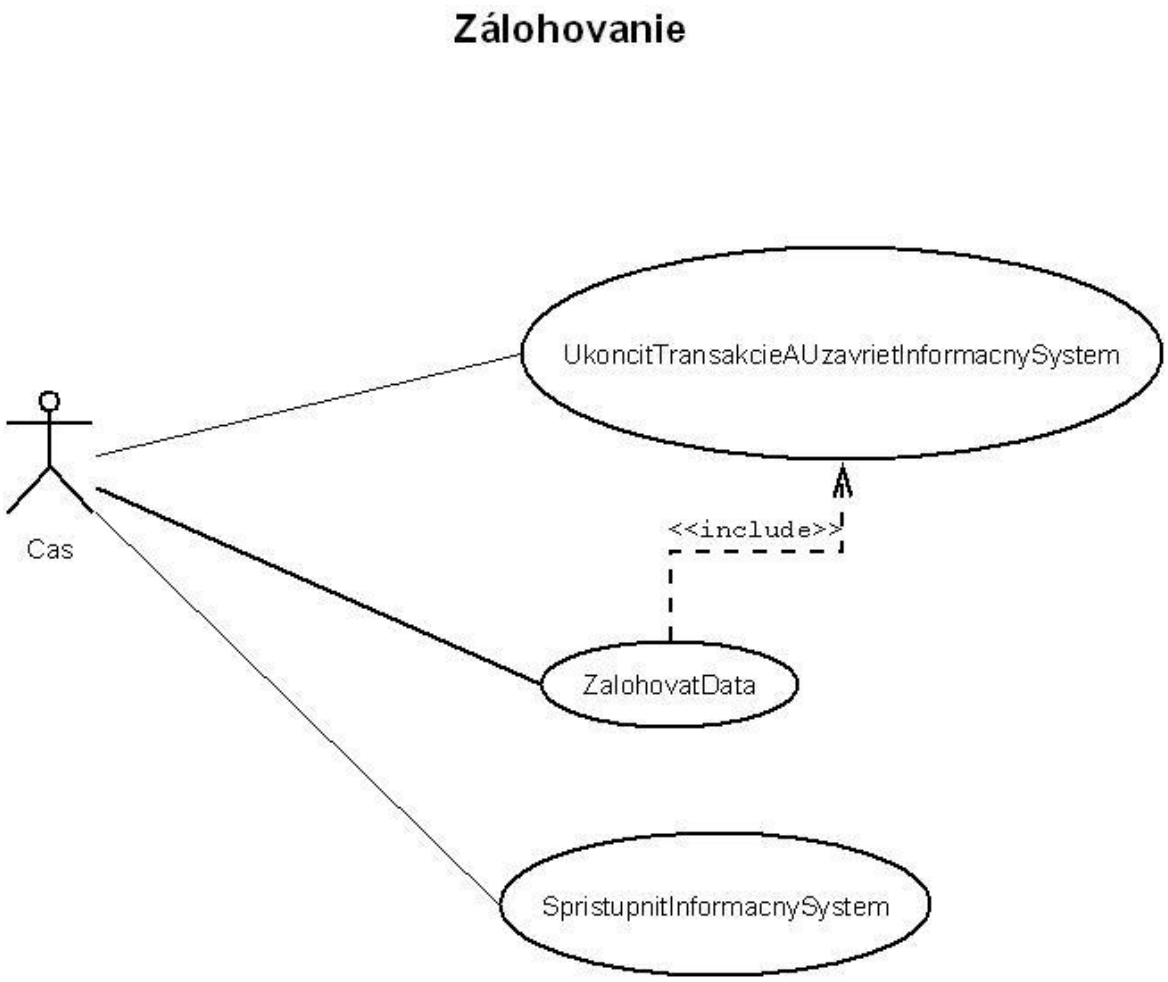
Zákazník je v našom prípade záujemca o lístok do divadla. Aby si mohol rezervovať lístok, môže využiť jednu z dvoch alternatív. Rezervovať si lístok priamo v pokladni divadla, alebo rezervovať si lístok cez webové rozhranie po prihlásení. Lístok bude možné vyplatiť priamo v divadle, alebo bankovým prevodom. Zákazník si lístok môže vyzdvihnúť po predložení osobných údajov priamo v pokladni divadla.



Obr. 22 – Rezervácia lístkov.

10.3.4 Zálohovanie dát

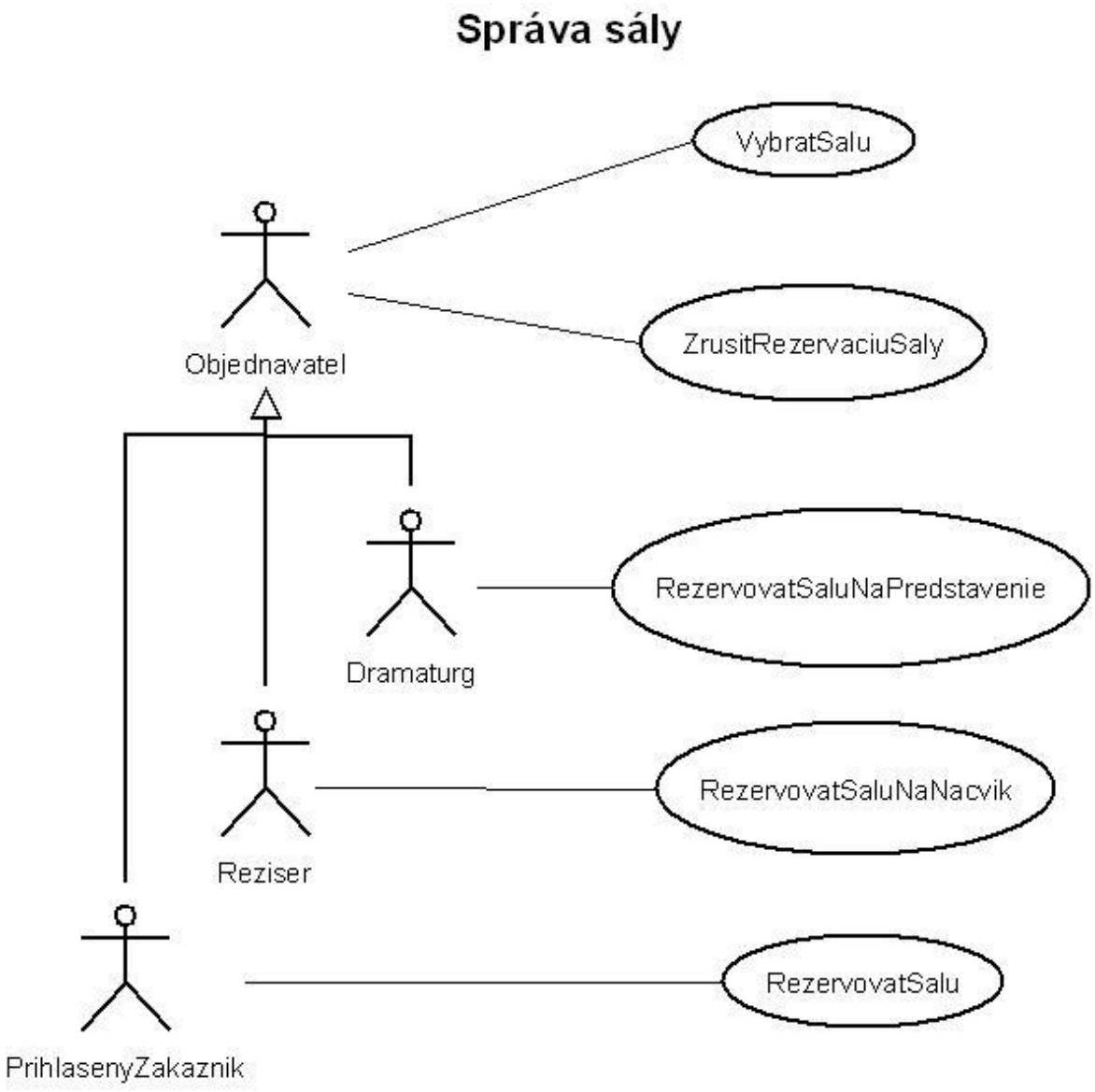
Aby divadlo poskytovalo pre zákazníka čo najlepšie služby a zákazníkov si udržalo čo najdlhšie, je potrebné aby sa zákazníkovi nestalo, že sa jeho rezervácia lístku stratila. Takisto by utrpelo aj celé divadlo keby sa informácie o nácvikoch a hrách stratia. Preto sa každý deň o 04:00 budú všetky informácie zálohovať. I keď je málo pravdepodobné, že v tento čas sa bude niečo meniť, je vhodné všetky transakcie ukončiť a uzavrieť celý systém. Potom sa dáta môžu zálohovať a celý systém sa po cca 10 minútach znova sprístupní.



Obr. 23 – Zálohovanie dát.

10.3.5 Správa sály

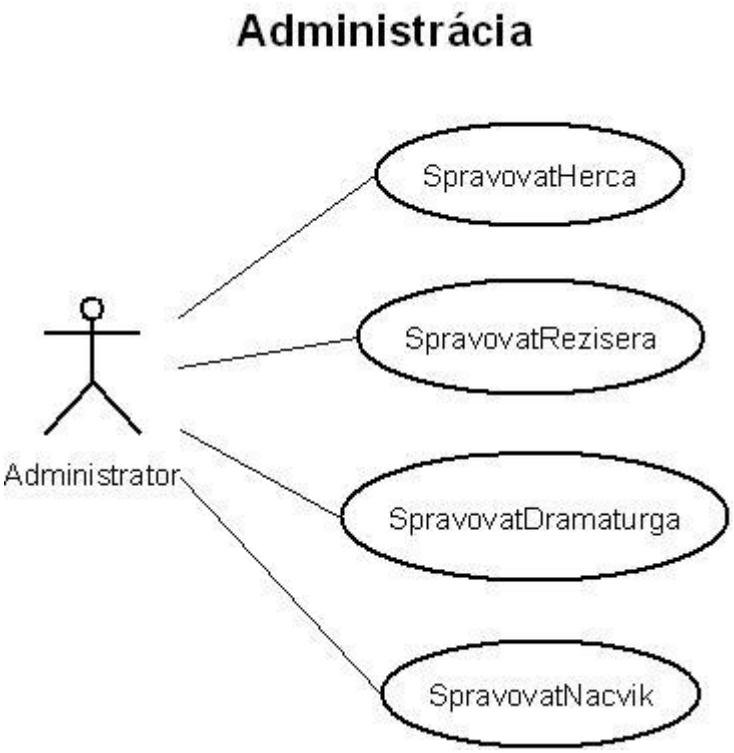
Dramaturg, režisér, alebo zákazník majú možnosť rezervovať si priestor v divadle. Dramaturg má možnosť rezervovať priestory pre predstavenie, režisér si môže rezervovať priestory na nácvik predstavenia a zákazník (fyzická alebo právnická osoba) si môže priestory divadla rezervovať pre vlastné potreby (rôzne spoločenské/charitatívne akcie atd.).



Obr. 24 – Správa sály.

10.3.6 Administrácia

Administrátor je aktér, ktorý má plný prístup k celému informačnému systému. V prípade potreby môže spravovať všetky údaje. V modeli sa uvádzajú len niektoré.

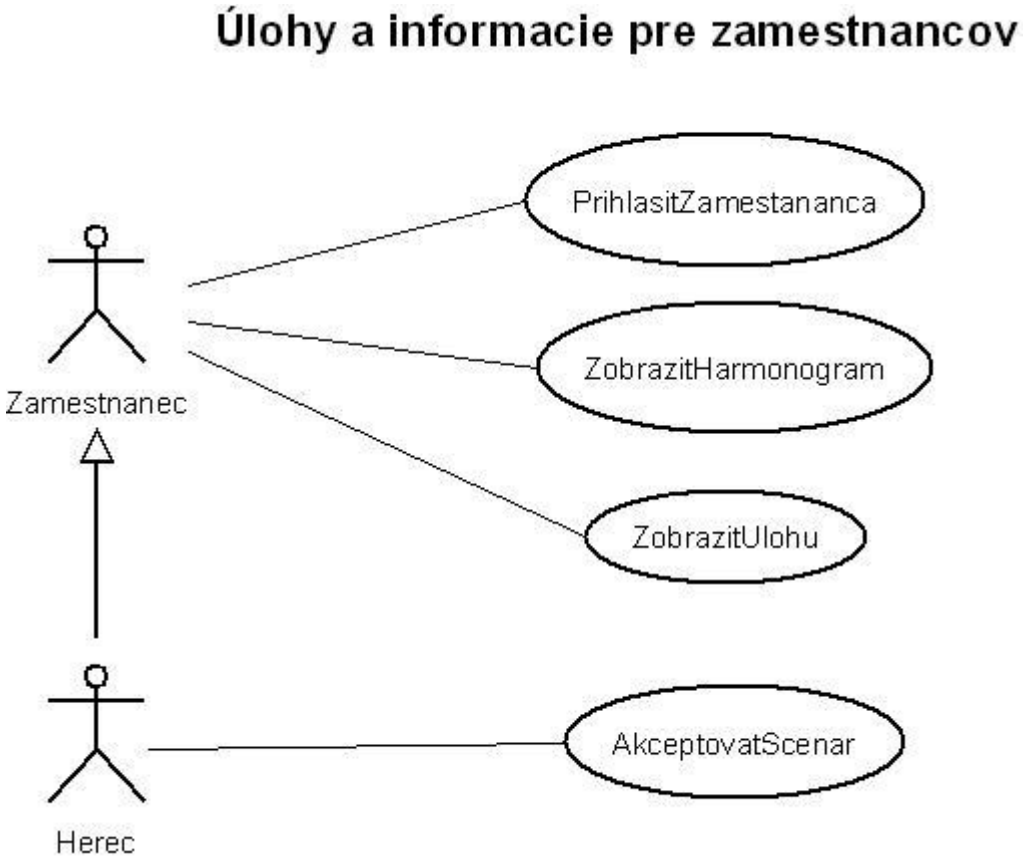


Obr. 25 – Administrácia.

10.3.7 Úlohy a informácie pre zamestnancov

K informačnému systému divadla budú pristupovať aj zamestnanci. Môžu si zobraziť svoje úlohy, svoj harmonogram a nájdu tam pre seba informácie kde a kedy majú vykonať určitú úlohu.

V prípade nesúhlasu herca so svojou úlohou v danej hre, prípadne, že sa mu celá hra nepáči, môže takúto úlohu herec zamietnuť.



Obr. 26 – Úlohy a informácie pre zamestnancov.

10.4 Slovné scenáre

Use Case : ZaevidovatHru
ID : UC1
Opis : Dramaturg pridá do systému novú alebo už predtým hranú hru
Primárny aktér : Dramaturg
Sekundárny aktér :
Vstupné podmienky : 1.Manažment si vyberie určitú hru 2.Nakúpenie autorských práv
Hlavný tok : 1.Dramaturg vyhľadá hru v databáze divadla 2.Ak hra nie je v systéme, dramaturg ju zaeviduje (zadá autora , názov hry a dobu hrania) do systému ako novú hru 3.Dramaturg zistí v akom jazyku je hra napísaná <PrelozitHru> 4. Zahrnúť (VybratRezisera) 5.Systém hru uloží do databázy divadla 6.Systém pridá hru do zoznamu hier pripravených na nacvičovania
Výstupné podmienky:
Alternatívny tok : 2.a. Ak sa v divadle táto hra už hrala, dramaturg iba pridá ďalšie obdobie hrania

Use Case : ZaevidovatHerca
ID : 2
Opis : Zaevidovanie herca do informačného systému režisérom
Primárny aktér : Režisér
Sekundárny aktér : Herec
Vstupné podmienky : 1.Záujem režiséra o herca 2.Podpísanie zmluvy herca s divadlom
Hlavný tok : 1.Režisér zaeviduje herca do systému (zadá meno, priezvisko, hru v ktorej hrá a jeho rolu v tejto hre) 2.Systém uloží herca do databázy hercov
Výstupné podmienky:
Alternatívny tok :

Use Case : VyberPredstavenia
ID : 3
Opis : Výber jedného predstavenia z ponuky predstavení patriacich k divadelnej hre
Primárny aktér : Prihlásený Zákazník , Neprihlásený Zákazník
Sekundárny aktér :
Vstupné podmienky : 1.Zákazník má už vybrané divadelné predstavenie
Hlavný tok : 1.Zahrnúť (VybratHru) 2.Systém zistí či hra má naplánované predstavenia 3.Systém ponúkne zákazníkovi zoznam všetkých ešte nehraných predstavení. 4.Zákazník si vyberie jedno z ponúknutých predstavení
Výstupné podmienky:
Alternatívny tok :

Use Case : ZalohovatData
ID : 4
Opis : Každodenná akcia pre zálohovanie všetkých dát
Primárny aktér : Čas
Sekundárny aktér :
Vstupné podmienky : 1.Čas nadobudol požadovanú hodnotu 4:00
Hlavný tok : 1.Zahrnúť (UkoncitTransakcieAUzamknutInformacnySystem) 2.V čaše o 4:00 sa systém automaticky zálohuje 3.Všetky informácie sa skopírujú na zálohovacie médium
Výstupné podmienky: 1. Úspešne vytvorená záloha systému
Alternatívny tok :

Use Case : ZrusitRezervaciu
ID : 5
Opis : Zrušenie rezervovaného priestoru v divadle
Primárny aktér : Objednávateľ
Sekundárny aktér : Dramaturg , Režisér , Prihlásený Zákazník
Vstupné podmienky : 1.Existuje rezervácia sály 2.Záujem o zrušenie rezervácie sály
Hlavný tok : 1.Používateľ sa rozhodne zrušiť rezervovaný priestor v divadle. 2.Systém zobrazí používateľovi všetky aktuálne rezervované priestory. 3.Pouižívateľ si vyberie jeden alebo viac z rezervovaných priestorov. 4.Systém zobrazí všetky kontrolne údaje 5. Ak je používateľ (sekundárny aktér) prihlásený zákazník zobrazí upozornenie o pokute 6.Používateľ potvrdí správnosť údajov. 7.Systém vykoná zrušenie rezervácie.
Výstupné podmienky: 1.Rezervácia sály je zrušená
Alternatívny tok :

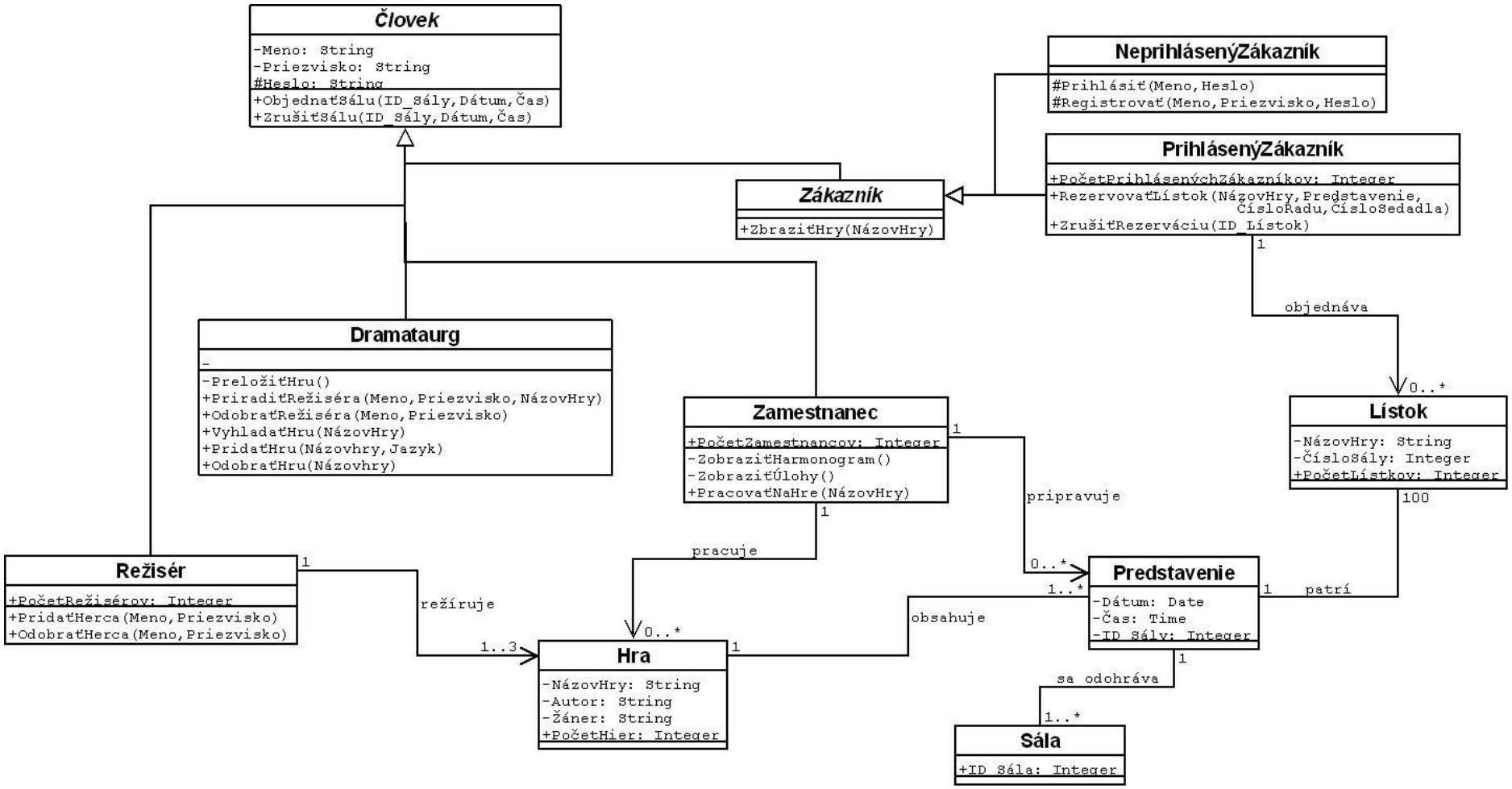
Use Case : SpravovatRezisera
ID : 6
Opis : Možnosť administrátora kde môže pridať, odstrániť, aktualizovať alebo čítať informácie o režisérovi
Primárny aktér : Administrátor
Sekundárny aktér :
Vstupné podmienky : 1. K systému je prihlásený administrátor
Hlavný tok : 1.Administrátor vyhľadá režiséra 2. Systém zobrazí režisérov podľa vyhľadávacích kritérií 3. Ak Administrátor zvolí "vymazať režiséra", 3.1 Systém odstráni režiséra so systému 4. Ak Administrátor zvoli "čítať režiséra", 4.1 Systém zobrazí Administrátorovi údaje o režisérovi 5. Ak Administrátor zvoli "aktualizovať režiséra", 5.1 Systém zobrazí formulár v ktorom môže administrátor údaje o režisérovi meniť 6. Ak kritériám nevyhovuje žiadny režisér 6.1 Systém oznámi Administrátorovi, že hľadaného režiséra nenašiel 6.2 Systém ponúkne administrátorovi voľbu "vytvoriť režiséra" 6.3 Ak administrátor zvoli "vytvoriť režiséra" zobrazí sa mu formulár na vytvorenie nového režiséra
Výstupné podmienky:
Alternatívny tok :

Use Case : ZobrazitHarmonogram
ID : 7
Opis : Daná možnosť zobrazí zamestnancovi jeho pracovný harmonogram aj s náhľadom na úlohu v daný pracovný deň
Primárny aktér : Zamestnanec
Sekundárny aktér :
Vstupné podmienky : 1. K systému je prihlásený zamestnanec
Hlavný tok : 1. K sýtemu sa prihlási zamestnanec 2. Zvolí možnosť zobraziť harmonogram 3. Systém zobrazí zamestnancovi celomesačný harmonogram <ZobrazitUlohu>
Výstupné podmienky:
Alternatívny tok :

10.5 Matica sledovateľnosti požiadaviek

	ZaevidovatHru	PrelozitHru	VybratRezisera	ZmenitObdobieHrania	VyhladatHru	ZaevidovatHerca	SpravaovatNacvik	PridatHercaNaNacvik	PridatHercaNaPredstavenie	IfmormovatZamestnanaca	RegistrovatZakaznika	PrihlasitZakaznika	VyhladatHru	VybratHru	VybratPredstavenie	VybratMiestoNaSedenie	RezervovatMiesto	ZrusitMiesto	UkoncitTransakcieAUzavrietSystem	ZalohovatData	SpristupnitInformacnySystem	VybratSalu	ZrusitRezervaciuSaly	RezervovatSalu	PrihlasitZamestnanca	ZobrazitHarmonogram	ZobrazitUlohu	OdmietnutScenar
1							**X**																					
2								**X**																				
3									**X**																			
4							**X**																					
5	**X**			**X**	**X**								**X**	**X**	**X**													
6																**X**	**X**	**X**										
7						**X**				**X**	**X**	**X**										**X**	**X**	**X**	**X**	**X**	**X**	**X**
8			**X**																									
9		**X**																										
10																			**X**	**X**	**X**							

10.6 Diagram analytických tried



Obr. 27 – Diagram analytických tried.

Záver

Tvorba a návrh informačných systémov je prostriedok, ktorý nám uľahčuje predovšetkým „papierovú“ prácu a dáva našim dokumentom tretí rozmer. Už nemusíme dlhé hodiny hľadať medzi hromadou papierov či ten a či onen produkt je na sklade, či si v kine, alebo divadle nájdeme voľné sedadlo, ani zamestnávateľ nemusí dlho do noci sedieť za stolom s kalkulačkou, či jeho zamestnanci dodržali normy. Na toto všetko nám stačí pár kliknutí myšou. Taktiež nemusíme mať pre naše dokumenty vytvorenú „obývaciu izbu“, keďže v počítači nám zaberú len zopár Kb a pritom sú nám kedykoľvek bez zdĺhavého hľadania k dispozícii.

Vývoj informačného systému nie je jednoduchá záležitosť a keďže spravidla nejde o sériovú výrobu, tento zložitý proces vyžaduje veľké investície času, finančných prostriedkov, úsilia a kreatívnych nápadov. V súčasnej dobe sú informačné systémy veľmi sofistikované a vytvárané „na mieru“ podľa požiadaviek individuálneho zákazníka. Preto do tohto zložitého procesu kreovania informačného systému vstupuje relatívne nový faktor, a to potreba komunikácie potrieb zákazníka, individuálny prístup k jeho problému a hlavne spätná väzba. Napriek snahe neopomenúť žiadny detail, často krát môže naše snaženie skončiť neúspechom.

Z tohto dôvodu, minimalizovania rizika neúspechu, s ktorým sa stretali programátori v minulosti boli vyvinuté a časom zdokonalené rôzne prístupy vývoja informačných systémov.

Preto sme pre lepšie pochopenie danej problematiky v úvodných kapitolách bakalárskej práce definovali pojmy ako softvérové inžinierstvo, informačný systém a rozviedli problematiku jeho vývoja a životného cyklu.

Pre splnenie nami stanoveného cieľa sme sa v nasledujúcich kapitolách zaoberali teoretickým rozborom dátového, procesného modelovania a jazyku UML, ktorý sme využili v poslednej kapitole práce .

V poslednej kapitole bakalárskej práce sme vytvorili návrh informačného systému pre divadlo. Na základe funkčných a nefunkčných požiadaviek sme pre aktérov vytvorili prípady použitia. Následne sme k nim vytvorili slovné scenáre a nakoniec diagram analytických tried.

Zoznam použitej literatúry

1. KADLEC,V. Agilní programování. Brno: Computer Press, 2004. 278 s. ISBN 80-251-0342-0

2. SIM, L. Princípy informačných systémov [online]. 2005 [cit.2010-05-06]. Dostupné na Internete: <http://rouen.yweb.sk/fiit/data/3_sem/pis/20051020_def.pdf>

3. Čo je informačný systém [online]. [cit.2010-05-07]. Dostupné na Internete: <http://edu.ukf.sk/mod/resource/view.php?id=25982>

4. ITEI_06_CASE_prostredky_v2 [online]. 2009 [cit.2010-05-07]. Dostupné na Internete: <http://im2-fim-uhk.wikispaces.com/file/view/ITEI_06_CASE_prostredky_v2.doc>

5. KOVÁČ, J. Rozdielnosť prístupu k testovaniu v tradičných a agilných metodikách [online].2010 [cit.2010-05-08]. Dostupné na Internete: <http://www2.fiit.stuba.sk/~bielik/courses/msi-slov/kniha/2010/essays/Kovac-msipapersource100.pdf>

6. BEŠENYI, L. Budovanie informačných systémov I. [online]. 2006 [cit.2010-05-08] http://www.inet.sk/clanok/3741/budovanie-informacnych-systemov-i-zivotny-cyklus-a-instalacia-mysql

7. ŽELEZNÍK, O. Podnikové informačné systémy a databázy [online]. 2008 [cit.2010-05-08]. Dostupné na Internete: <http://files.riadeniekvality.webnode.sk/200000700-53bf955b3d/PIS.pdf>

8. Životný cyklus [online]. [cit.2010-05-09] Dostupné na Internete: <http://edu.ukf.sk/mod/resource/view.php?id=25989>

9. Internet [online]. [cit.2010-05-09]. Dostupné na Internete: <http://sk.wikipedia.org/wiki/Rational_Unified_Process>

10. Základné prvky a vlastnosti objektového prístupu, ich notácia a implementácia [online]. [cit.2010-05-09] Dostupné na Internete: <http://www.gratex.com/Download/OOANS02TechDiag2.pdf>

11. Procesné modelovanie [online]. [cit.2010-05-10]. Dostupné na Internete: <http://www.softec.sk/showdoc.do?docid=951>

12. KANISOVÁ, H. – MÜLLER, M. UML srozumitelne. 2. Vyd. Brno: Computer Press, 2006. 176 s. ISBN 80-251-1083-4

13. Databázové systémy 1 [online]. 2007 [cit.2010-05-10]. Dostupné na Internete: < hornad.fei.tuke.sk/predmety/dbs/dbs-pr/pr1.pps>

14. ZELENKA, P. WebML - datové modelování [online]. 2004 [cit.2010-05-10]. Dostupné na Internete: <http://interval.cz/clanky/webml-datove-modelovani/>

15. Internet [online]. [cit.2010-05-11]. Dostupné na Internete: < http://cs.wikipedia.org/wiki/CASE_n%C3%A1stroje>

16. ARLOW, J. – NEUSTADT, I. UML a unifikovaný proces vývoje aplikací. Brno: CP Books, a.s., 2005. 387 s. ISBN 80-7226-947-X

17. Analýza zadania, [online]. [cit.2010-05-11] Dostupné na Internete: <http://edu.ukf.sk/mod/resource/view.php?id=25992>

18. ADAMUŠČINOVÁ ,I. Diagram prípadov použitia [online]. 2008 [cit.2010-05-11] Dostupné na Internete : <s.cnl.tuke.sk/~ywetka/USE_CASE.ppt>

19. Slovník pojmov [online]. [cit.2010-05-17]. Dostupné na Internete: <http://www.dynatech.sk/informacne-systemy/technologie/uml.aspx>

20. Internet [online]. [cit.2010-05-18]. Dostupné na Internete: <http://era.nih.gov/docs/rup_fundamentals_slide03.jpg>

21. Pajbach, M. Dôležitosť manažmentu rizík v malých softvérových projektoch [online]. 2009 [cit.2010-05-18]. Dostupné na Internete: <http://www2.fiit.stuba.sk/~bielik/courses/msi-slov/kniha/2010/essays/Pajbach-msipapersource15.pdf>

22. Oláh, M. Agilné plánovanie [online]. 2008 [cit.2010-05-18]. Dostupné na Internete: <http://www2.fiit.stuba.sk/~bielik/courses/msi-slov/kniha/2009/essays/msipaper067-olah.pdf>

23. Internet [online]. [cit.2010-05-23]. Dostupné na Internete: <http://cs.wikipedia.org/wiki/Vodop%C3%A1dov%C3%BD_model>

24. Mederly, P. Softvér a proces vývoja softvéru [online]. 1998 [cit.2010-05-23]. Dostupné na Internete: <cyril.fmph.uniba.sk/www/sw-eng/99/p0.doc>

25. Nagy, P. Návrh a projektovanie IS [online]. 2002 [cit.2010-05-23]. Dostupné na Internete: <http://fel.uniza.sk/~nagy/IS/PDF/IS_K8.pdf>

Prílohy

Príloha – CD obsahujúce pdf verziu práce