VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

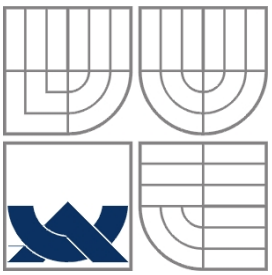
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

INFORMAČNÍ SYSTÉM PRO BAROVÁ A RESTAURAČNÍ ZAŘÍZENÍ

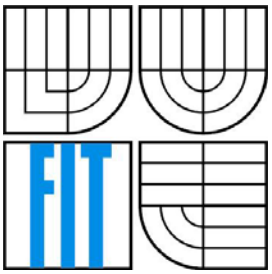
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE Bc. Petr Mašlaň
AUTHOR

BRNO 2008



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

INFORMAČNÍ SYSTÉM PRO BAROVÁ A RESTAURAČNÍ ZAŘÍZENÍ

INFORMATION SYSTÉM FOR RESTAURANTS AND BARS

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

Bc. PETR MAŠLAŇ

VEDOUCÍ PRÁCE
SUPERVISOR

Ing. Ivana Rudolfová

BRNO 2008

Abstrakt

Cílem práce je prostudovat metody určování požadavků na software a metody specifikace a vytvořit informační systém na platformě .NET, který se bude zaměřovat na restaurační a barová zařízení. Informační systém bude tedy zpracovávat a archivovat informace o stavu zboží, nasazování zaměstnanců do směn, přehled příjmů a výdajů a knihu norem.

Klíčová slova

Restaurace, bary, IS, .NET, C#, určování požadavků, specifikace požadavků, rozpis směn, kniha norem, LINQ, SQL, uložená procedura

Abstract

The aim of this work is to study methods for requirement determination and specification methods to create information system on the .NET platform which will be specialized for restaurant and bar establishments. The information system will be therefore process and archive informations about condition of goods, employee's schedule of working shifts, summary of incomes and expenses and book of norms.

Keywords

Restaurants, bars, IS, .NET, C#, determination of requirements, specifications of requirements, LINQ, SQL, Stored Procedure

Informační systém pro barová a restaurační zařízení

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod Ing. Ivany Rudolfové.
Další cenné informace o provozu a softwarových potřebách v jeho barech a zařízeních mi poskytl Jaroslav Mašlaň.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
Petr Mašlaň
16.5.2008

Poděkování

Dovoluji si poděkovat vedoucí diplomové práce pani Ing. Ivaně Rudolfové za odborné vedení a mnoho cenných připomínek k předložené práci. V neposlední řadě je moji milou povinností poděkovat za nevšední podporu rodičům a celé rodině při studiu a zároveň také kolegům, kteří užitečnými radami pomohli k vypracování této diplomové práce. Děkuji také panu RNDr. Petru Pahutovi, DrSc. za cenné rady a připomínky k této diplomové práci.

Obsah

Obsah 1
1 Úvod 4
2 Softwarový proces a životní cyklus softwaru 5
2.1 Iterativní a inkrementální proces 6
2.2 Model zralosti (CMM) 6
2.3 Etapy životního cyklu softwaru 8
2.3.1 Analýza a specifikace požadavků 8
2.3.2 Architektonický návrh 8
2.3.3 Podrobný návrh 9
2.3.4 Implementace součástí 9
2.3.5 Integrace a testování systému 9
2.3.6 Akceptační testování a instalace 9
2.3.7 Provoz a údržba 9
2.4 Modely životního cyklu softwaru 9
2.4.1 Vodopádový model 10
2.4.2 Iterativní model 10
2.4.3 Inkrementální model 11
2.4.4 Spirálový model 11
2.4.5 Rational Unified Process 11
2.4.6 Agilní metodologie 12
2.5 Životní cyklus databáze 12
3 Analýza a specifikace požadavků 14
3.1 Získání, analýza a definování požadavků 14
3.1.1 Transformace neformálních požadavků do strukturované podoby 14
3.1.2 Provedení studie vhodnost, identifikace a analýza rizik 14
3.1.3 Plánování akceptačního testování 14
3.2 Určování požadavků 15
3.2.1 Funkční a nefunkční požadavky 15
3.3 Získání požadavků 16
3.3.1 Tradiční metody získání požadavků 17
3.3.2 Moderní metody získávání požadavků 19
3.3.3 Uspořádání a validace požadavků 22
4 Analýza 23
4.1 Databáze 23

4.1.1 Relační databáze 24
4.1.2 SQL (Structured Query Language) 25
4.1.3 T-SQL (Transact SQL) 25
4.1.4 Uložené procedury 25
4.2 Microsoft Visual Studio .NET 26
4.3 Jazyk C# 28
4.3.1 Kolekce .NET 3.5 28
4.3.2 Nová klíčová slova C# 3.0 29
4.4 LINQ 29
4.4.1 LINQ to SQL 30
4.4.2 Mapování LINQ to SQL 30
4.5 Překlad LINQ na SQL 31
4.5.1 Mapování databáze do objektů 32
4.6 Informační systémy 33
4.7 Microsoft .NET 34
4.7.1 Microsoft .NET Framework 35
5 Analýza informačního systému pro restaurační a barová zařízení 39
5.1 Požadavky 39
5.1.1 Evidence zboží 39
5.1.2 Peněžní deník 40
5.1.3 Kniha norem 40
5.1.4 Rozpis směn 41
5.2 Stávající systém 41
5.2.1 Evidence zboží 41
5.2.2 Peněžní deník 41
5.2.3 Kniha norem 42
5.2.4 Rozpis směn 42
5.3 Neformální specifikace 42
5.3.1 Rozpis 42
5.3.2 Peněžní deník 42
5.3.3 Kniha norem 42
5.3.4 Rozpis směn 43
5.3.5 Uživatelské účty 43
5.4 Diagramy případu užití 43
5.5 ER diagramy 47
5.5.1 Rozpis 47
5.5.2 Peněžní deník 48

5.5.3 Rozpis směn 49
5.5.4 Kniha norem 50
5.5.5 Uživatelská práva 50
6 Návrh řešení 52
6.1 Návrh architektury 52
6.2 Návrh databáze 53
7 Implementace 54
7.1 Výběr databázového systému 54
7.2 Výběr operačního systému 54
7.3 Výběr vývojového prostředí 55
7.4 Návrh uživatelského rozhraní 55
7.5 Struktura aplikace 56
7.5.1 Evidence zboží 56
7.5.2 Peněžní deník 57
7.5.3 Rozpis směn 57
7.5.4 Kniha norem 57
7.5.5 Tiskový modul 58
8 Závěr 59
Literatura 60
Příloha 61

1 Úvod

Cílem této diplomové práce je prostudovat metody určování požadavků na software a následně je aplikovat na reálný informační systém. Tedy použít vhodné metody určování požadavků na software a tím získat požadavky na informační systém. Následně získané požadavky převést na vhodnou formu, nejlépe takovou, aby byla čitelná a srozumitelná jak ze strany zadavatele, tak i ze strany vývojáře. Touto fázi se zabývá analýza.

V této práci se budu v dalších částech zabývat softwarovým procesem a jeho vývojem, tj. etapami životního cyklu softwaru. Další část je věnována metodám určování požadavků (jak tradičním, tak moderním) a jejich následné analýze a zpracování.

Následující část diplomové práce je věnována aplikování metod určování požadavků, jejich analýze a vytvoření potřebným modelů. Kapitola analýza je zaměřena na technologie, které budou použity ve fázi implementace. Jedná se především o vývojové prostředí, programovací jazyk a jeho možnosti a schopnosti databázového systému.

Z vytvořených modelů bude vypracován fyzický model databáze a potřebné uložené procedury pro manipulaci s daty. V následujícím kroku bude vytvořena aplikaci na platformě .NET, které bude schopna komunikovat s databází pomocí vytvořených uložených procedur. Pro efektivnější komunikaci mezi databází a aplikací se vytvoří datové struktury LINQ, které budou uchovávat jen potřebná data na straně klienta a umožní synchronizaci s databází.

Aplikace umožní uživatelům snazší vedení denní kancelářské práce s množstvím rozšíření a její elektronickou archivaci. Uživatelé získají možnost analyzovat prodeje zboží za dané období.

Na závěr zhodnotím celý systém a dosažené výsledky.

2 Softwarový proces a životní cyklus softwaru

Tato kapitola je v první části orientována na vývoj softwarového procesu a jeho modely. Druhá část se zabývá životním cyklem softwaru.

Softwarový proces definuje, kdy, kdo a co má dělat, aby bylo dosaženo požadovaného cíle. Při vývoji softwaru vstupují do hry tři strany:

- Zákazník
- Dodavatel
- Uživatelé

Softwarový proces označuje aktivity a organizační procedury užité při softwarové výrobě a údržbě. Proces napomáhá ke správě a zlepšení spolupráce ve vývoji a udržování týmu, takže kvalitní produkt je dodán zákazníkům a následně je zajištěna jeho pozdější podpora. Model procesu zahrnuje:

- Stavy objednávky pro provedení činnosti
- Určení jaké vývojové prostředky jsou použity a kdy
- Přiřazení aktivit vývojářům
- Kritéria pro sledování stavu projektu, hodnocení výsledků a pro plánování budoucích projektů

Na rozdíl od modelování a programovacích jazyků nejsou softwarové procesy normalizovány. Každá organizace vyvíjí vlastní model procesů nebo si přizpůsobí model z obecné šablony, jako je Rational Unified Process.

Proces přijatý organizací musí být orientován na vývojářovy znalosti a dovednosti, manažerské praktiky, zákazníkovy požadavky, velikost projektu, atp. Protože jsou všechny tyto faktory předmětem změny, organizace může vyžadovat různé modely procesů a vytvářet tak odlišné varianty pro každý projekt.

Velikost projektu má největší vliv na proces. V malých projektech je přednější neformální komunikace v týmu. Avšak ve větších projektech neformální komunikace nepostačuje a je potřeba definovat, jakým způsobem budou mezi sebou jednotlivé skupiny vývojářů komunikovat. [7]

2.1 Iterativní a inkrementální proces

Moderní procesy vývoje softwaru jsou obvykle iterativní a inkrementální. Systémové modely jsou upravovány a transformovány při analýze, návrhu a implementačních fázích. Podrobnosti jsou přidávány v postupných iteracích, změny a vylepšení jsou přidávány podle potřeby. Inkrementální vydávání softwarových modulů poskytuje zpětnou vazbu modulů ve stádiu vývoje. Existuje několik variant iterativního a inkrementálního procesu:

- Spirálový model
- Rational Unuified Model (RUP)
- Model-driven architecture (MDA)
- Agilní vývojový proces

Spirálový model byl definován Boehmem v roce 1988, slouží jako referenční bod pro modernější modely, včetně tří modelů udaných výše. RUP je relativně pružný proces, který nabízí podporu k vedení vývojářů s různými šablonami, vysvětlivek návrhů, vývojových myšlenek. MDA je založen na proveditelných specifikacích – vytváří software z modelů a komponent. Agilní vývojový návrh upřednostňuje osoby a týmovou spolupráci před plánováním, dokumentací a jinými formalitami.

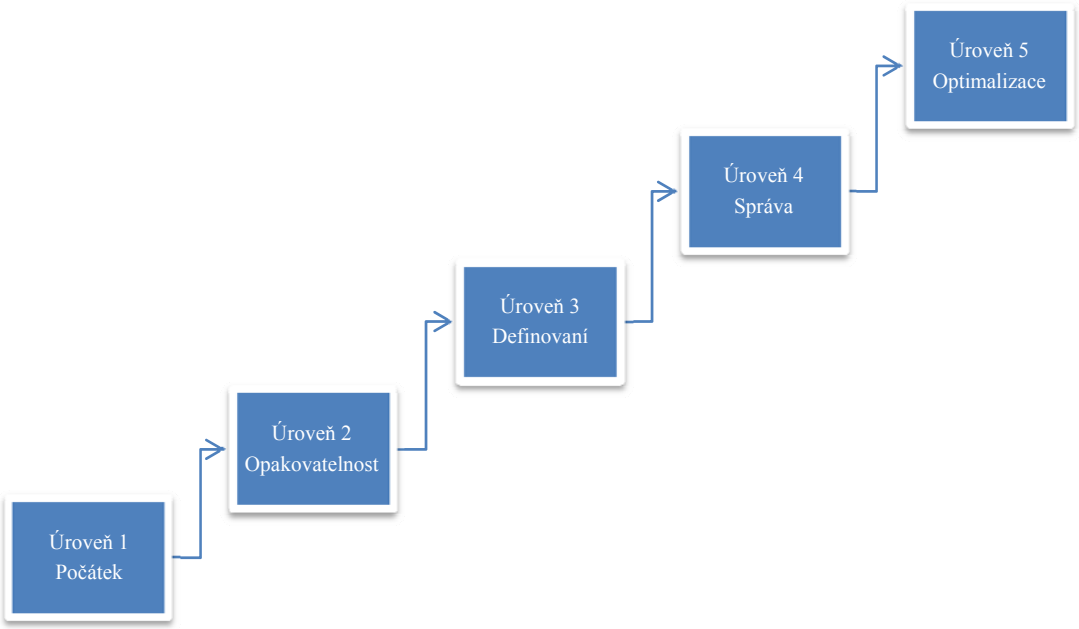
Stavy RUP: iterační proces je takový proces, který zahrnuje plánování dokončení jednotlivých spustitelných vydání. Inkrementální proces je takový proces, který zahrnuje spojitou integraci systémové architektury k vytvoření daných vydání, každé nové vydání zahrnuje vylepšení.

Jestliže iterativní a inkrementální vývoj je plánován a řízen, proces může degenerovat na proces bez řízení a projekt tak nemá skutečné pokroky. [7]

2.2 Model zralosti (CMM)

Hlavní výzvou pro každou organizaci zabývající se softwarem je zdokonalení vývojového procesu. Vývojový proces je dostatečně přirozený na to, aby se zahájil proces inovace. Organizace musí znát, jaké problémy jsou se současným procesem. Tento model zralosti CMM se využívá pro hodnocení a zlepšení stávajícího procesu.

CMM (Capability Maturity Model) je v zásadě dotazník, který vyplňuje IT organizace. Dotazník se následně verifikuje a ověřuje, aby se zařadil do jedné z pěti úrovní CMM. Nejvyšší úroveň má nejvyšší úroveň vyspělosti v organizaci.



Obrázek 2.1 model zralosti [6]

Úrovně modelu CMM:

1. Na této úrovni dominují nahodilé procesy. Systém je vytvářen bez firemních pravidel chaoticky, takže se jeho tvorba dostává do kritických situací. Dosažený úspěch ve vývoji software je důsledkem šťastné náhody a závisí na individuálních schopnostech programátorů. Dohodnuté termíny nejsou obvykle splňovány a ukončení vývoje je dosahováno heroickým úsilím na konci projektu. Celkové náklady jsou spočteny po ukončení vývoje a posuzují se jako důsledek vývoje a nutných výdajů. O kvalitě software se sice hovoří, ale nijak se nezajišťuje. Každý spoléhá na druhého, že kvalitu zajistí.
2. Opakovaně se dosahuje dobrých výsledků. Firma využívá základních postupů projektového řízení, ale z projektu na projekt se přenášejí jen některé úspěšné prvky řízení. Nicméně intuitivně zaběhané procesy a povědomí, že je potřeba pracovat kvalitně, vytvářejí dost stabilní prostředí pro udržení přijatelné úrovně jakosti softwarových produktů.
3. Software je vyvíjen podle předem definovaného postupu, metodicky, plánovitě, s využitím pokročilého projektového řízení s cílem dosáhnout vypracování požadovaného softwaru v čase, s rozpočtovanými náklady a disponibilními zdroji. Provádí se pravidelné vyhodnocování odchylek od plánu a přijímají se opatření ke krácení termínů jednotlivých činností, aby software byl dodán včas. O kvalitu produktů a služeb se s ohledem na zákazníky explicitně usiluje, proto je kvalita softwaru dodržována na velmi dobré úrovni a má tendenci vykazovat určité zlepšování.

4. Firma má všechny procesy jasně definovány a stanoven pro ně postup měření, který vyhodnocuje jejich podíl a efektivitu při tvorbě software. Zjištěné charakteristiky procesů jsou postupně upravovány tak, aby se firma přizpůsobila měnícím se podmínkám trhu, aniž by to mělo dopad na jakost vyvíjeného softwaru, jehož kvalitativní parametry jsou firmou cílevědomě stále zvyšovány a dosahují vysoké úrovně.
5. Neustálá zpětná vazba ovlivňuje následné softwarové projekty tak, aby se firemní procesy neustále zlepšovaly a dosahovaly předem definovaných, optimálních parametrů. Firma dosahuje trvale špičkové jakosti, aniž by náklady na kvalitu softwaru měly dopad na hospodaření firmy. Naopak, garantovaná kvalita software usnadňuje jeho prodej.

[6]

2.3 Etapy životního cyklu softwaru

Životní cyklus softwaru je důležitým aspektem při vývoji softwaru. Životní cyklus je rozdělen na několik etap, podle kterých se řídí vývoj softwaru od počátečních fází získávání požadavků až po nasazení do provozu. V kapitole se zmíníme také o některých základních modelech životního cyklu.

2.3.1 Analýza a specifikace požadavků

Analýza a specifikace požadavků je první etapou při vývoji softwaru. V této etapě se zabýváme požadavky zákazníka na výsledný produkt (software). Získáváme je, analyzujeme, definujeme a specifikujeme. Jinak řečeno, z nejasných, neformálních a mnohdy rozporuplných požadavků zákazníka se snažíme vytvořit strukturované, jasné a konzistentní požadavky. Analýza a specifikace požadavků se zabývá jen studií požadavků. Realizaci a testováním se zabývají další etapy životního cyklu.

Hlavním aspektem této etapy by měla být analýza proveditelnosti. Na základě této analýzy se zjistí, zda má smysl v projektu pokračovat, nebo jej zavrhnout a zákazníkovi nabídnout jiné řešení. Dalším výstupem této etapy může být zjištění možných rizik a jejich následná analýza. Důležitou součásti je plán akceptačního testování.

2.3.2 Architektonický návrh

Navazuje na předchozí etapu. Slouží k ujasnění koncepce systému a k jeho dekompozici na podsystémy. Dekompozice pak vyžaduje vymezení funkcionality jednotlivých podsystému a vztahů mezi nimi.

Při architektonickém návrhu se plánuje testování celého systému, což znamená testování, zda jednotlivé podsystémy správně komunikují a pracují v rámci jednoho celku. Je přinejmenším vhodné také naplánovat postup nasazení systému do provozu a zaškolení uživatelů.

2.3.3 Podrobný návrh

Má za úkol formálně popsat logickou a fyzickou strukturu dat, podrobný popis jednotlivých součásti a způsob ošetření chybových stavů a výjimek.

V tomto návrhu se také plánuji práce na implementaci jednotlivých součástí. S tím souvisí vytvoření požadavků na lidské zdroje a odhad doby trvání na projekt. Z tohoto lze spočíst náklady na vývojáře. Výstupem podrobného návrhu by měly být návrhy testů, včetně testovacích dat.

2.3.4 Implementace součástí

Jedná se o část, ve které se přenáší úsilí získané v předchozích částech do výsledného produktu. Zahrnuje programovou realizaci produktu, vytvoření programové dokumentace k jednotlivým součástem a následné otestování vytvořených součástí.

2.3.5 Integrace a testování systému

V této etapě spojíme součásti do jediného celku. Tento celek pak testujeme. Při tomto testování zjistíme chyby, které nemohly být odhaleny v předchozí fázi. Nalezené chyby opravíme, tedy vrátíme se k předchozí fázi.

2.3.6 Akceptační testování a instalace

Jedná se o testování, které provádí uživatel systému. Na základě této fáze se zákazník rozhodne, zda systém převezme, nebo ho pro závažnější nedostatky odloží, než je dodavatel opraví. Po akceptování systému zákazníkem je možné systém instalovat a zaškolit uživatele.

2.3.7 Provoz a údržba

Je nutné řešit problémy, které plynou s delším užíváním softwaru. Tato etapa také zahrnuje opravy vzniklých chyb, přidávání nových vlastností nebo přizpůsobování systému novým požadavkům. [7]

2.4 Modely životního cyklu softwaru

Každý softwarový projekt je unikátní, takže i jednotlivé etapy a jejich uspořádání se liší. Přesto existují rysy, které jsou společné, a můžeme je nalézt v modelu životního cyklu softwaru.

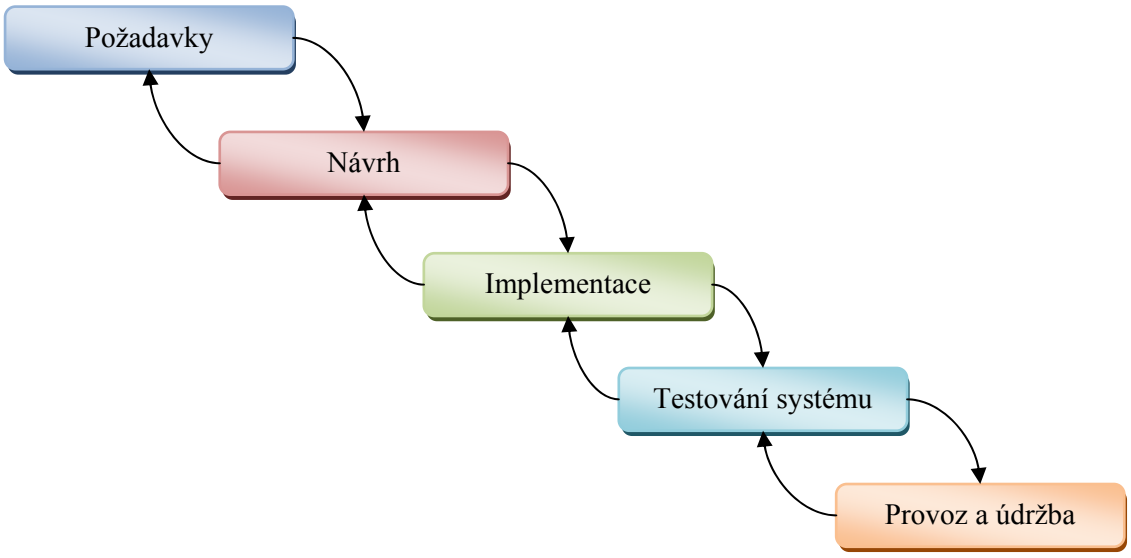
Model životního cyklu definuje jednotlivé etapy, které je potřeba vykonat včetně jejich časové návaznosti. Na konci každé etapy musí být vytvořen reálný výstup, aby bylo možné ověřit jeho správnost.

Existuje několik základních modelů životního cyklu softwaru. Mezi nejznámější patří vodopádový model, iterativní model, inkrementální model, spirálový model, dále Rational Unified Process (RUP) a agilní metodologie.

2.4.1 Vodopádový model

Vodopádový model je nejstarší model. Jednotlivé etapy modelu jsou seřazeny za sebou. Teprve po skončení jedné etapy může začít druhá. Nikdy nenastane situace, kdy se pracuje na dvou etapách zároveň.

Problémem vodopádového modelu je to, že uživatel není schopen předem definovat všechny požadavky na systém. V závěrečných fázích projektu se přináší nové požadavky a je potřeba vytvořit novou analýzu požadavků, upravit návrh, implementaci a testování. Uživatel vidí až finální verzi produktu, je zapojen jen v etapě požadavků.

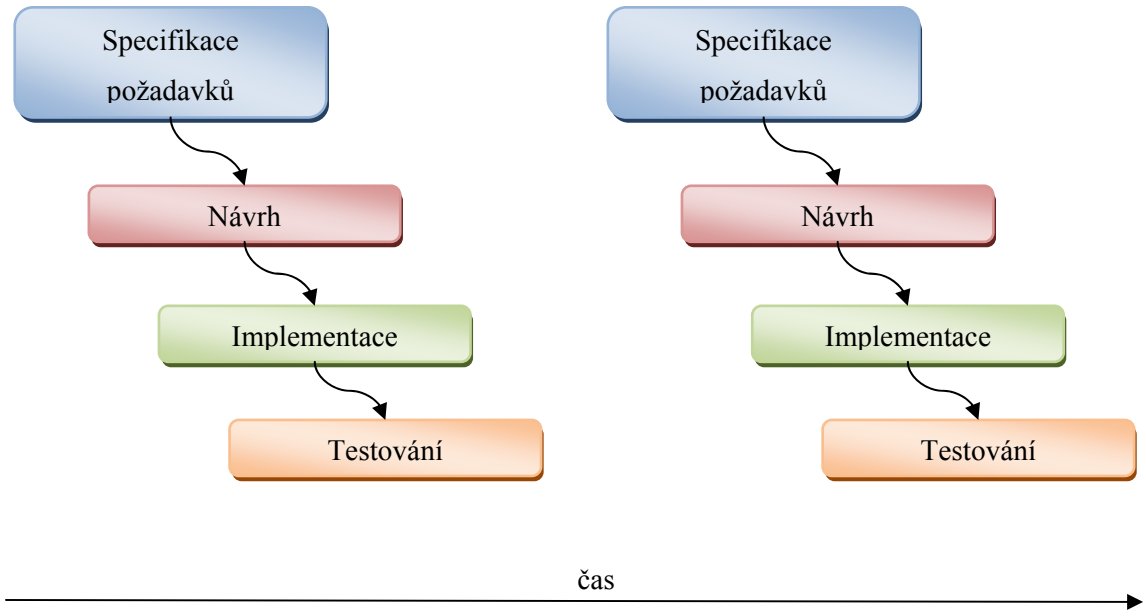


Obrázek 2.2 vodopádový model [7]

Z vodopádového modelu vycházejí ostatní modely životního cyklu. Vodopádový model lze použit i v praxi, kde je lepší použít přesně specifikovaný postup, než aby vznikl chaos. [7]

2.4.2 Iterativní model

Základní vlastností iterativní modelu je odstranění problému vodopádového modelu životního cyklu. Proces vývoje softwaru se rozdělí do iterací.



Obrázek 2.3 iterativní model [7]

Po každé iteraci získáme spustitelnou verzi softwaru, která napomáhá k upřesnění požadavků na výsledný software. Tyto požadavky pak budou zpracovány v další iteraci. Z toho plyne výhoda, že uživatel může v průběhu vývoje upřesňovat požadavky bez toho, aby narušil životní cyklus softwaru.

2.4.3 Inkrementální model

Model je podobný iterativnímu. Na základě specifikace systému je tvorba systému rozdělena do samostatných částí, které se postupně po vytvoření předávají uživateli.

2.4.4 Spirálový model

Model je založen na prototypování a analýze rizik. Jednotlivé kroky se ve spirále opakují, ale pokud možno ve vyšším stupni zvládnuté problematiky. Model se může zdát podobný inkrementálnímu nebo iterativnímu, uživatel ale nedostává k vyzkoušení verze softwaru. [7]

2.4.5 Rational Unified Process

Oproti předchozím modelům životního cyklu není RUP jen koncepcí, ale je použitelný pro řízení reálných softwarových projektů. RUP je rozsáhlá, propracovaná objektově orientovaná iterační metodologie vývoje softwaru. Důraz je kladen na vizualizaci softwarového systému, na průběžnou kontrolu kvality, na řízení změn a na využívání existujících komponent.

RUP zavádí čtyři základné fáze vývoje (zahájení, projektování, realizace, předání), kde každá je organizována do několika iterací. Před začátkem nové iterace musí být dokončeny předchozí iterace. [5]

2.4.6 Agilní metodologie

Metodologie založené na klasických modelech životního cyklu softwaru se vyvíjely dlouhou dobu. Při vytváření menších projektů pak nebyly příliš výhodné. Dodržování pravidel projekt zpomalilo a také prodražilo. Agilní metodologie kladou důraz na člověka jako určující faktor pro kvalitu výsledku.

Agilní programování je ucelený přístup k vývoji software, který se snaží rychle vyvinout, otestovat a nasadit verzi (iteraci) software u zákazníka. Podle reakcí a okamžitých potřeb zákazníka se potom ubírá vývoj další verze (iterace). Jednotlivé iterace jsou krátké časové úseky, typicky jeden až čtyři týdny a jsou to vlastně miniaturní softwarové projekty, které zahrnují všechny fáze běžné u standardních projektů: plánovaní, analýza, návrh, kódování, testování a dokumentace. Jedna iterace sice nepřidá příliš funkčnosti, ale je důležité, aby na konci každé iterace byla nová, funkční verze software. Na konci každé iterace vývojáři vyhodnocují a případně mění priority projektu. Agilní programování dává přednost komunikaci, nejlépe osobní, před psanou dokumentací. Týmy, které používají agilní techniky, jsou v neustálém kontaktu a většinou pracují i v jednom prostoru. To se týká minimálně programátorů a zákazníků. Důležitá je velikost týmu, komunikace nefunguje (vzniká příliš mnoho interakcí) pokud je v týmu více než 20-40 členů. Nejznámější odrůdou agilního programování je tzv. extremní programování, které klade důraz na párové programování, automatické texty, úzkou interakci vývojáře a zákazníka. [11]

2.5 Životní cyklus databáze

Životní cyklus databáze definuje úplný proces od konceptu k implementaci. Celý proces vývoje a implementace životního cyklu může být rozdělen na menší fáze.

Před začátkem vývoje každého systému je potřeba mít silný model životního cyklu. Model musí mít všechny fáze definovány ve vhodném pořadí tak, aby napomáhaly vývojovému týmu k vytvoření systému s požadovanými problémy a funkcionalitou.
Databázový životní cyklus zahrnuje následující etapy od základních kroků, zahrnující návrh a globální schéma databáze, po implementaci a údržbu:

- **Analýza požadavků:** Požadavky musí být určeny před procesem návrhu a implementace.
- **Logický návrh:** Po shromáždění požadavků, dat a relací musí být definován konceptuální datový model, jako je ER diagram (případně diagram tříd).

- **Fyzický návrh:** Jakmile je vytvořen logický návrh, následuje krok zahrnující vytvoření fyzického modelu databáze. Fáze fyzického návrhu zahrnuje vytvoření tabulek a zvolení indexů.
- **Databázová implementace:** Jakmile je návrh ukončen, může být vytvořena databáze z vytvořených návrhů pomocí DDL (Data Definition Language)
- **Editace dat:** DML (Data Modification Language) může být použit k dotazům a úpravám databáze, jakmile jsou nastaveny indexy a omezení jako je referenční integrita.
- **Monitorování databáze:** Po uvedení databáze do provozu monitorování zajistí sledování výkonových požadavků. Jestliže je výkon nedostačující, je potřeba databázi modifikovat, tudíž zahájit nový životní cyklus.

[12]

3 Analýza a specifikace požadavků

Ještě před tím, než začneme provádět objektově orientovanou analýzu, měli bychom mít alespoň rámcový přehled toho, co budeme tvořit a jaký je smysl požadavků a jejich specifikace. Musíme jednak zjistit, co vlastně bude systém dělat a jednak v tomto bodu dosáhnout shody.

V podstatě každý projekt může mít více druhů uživatelů, techniků údržby, pomocného personálu, prodavačů, manažerů apod. Inženýrství požadavků spočívá ve stanovení služeb, které by měl vyvíjený systém poskytovat, a omezení, za nichž musí pracovat. Spočívá rovněž v získání požadavků, jaké na nový systém mají jeho uživatelé, včetně respektování jejich priorit. Je to proces vyjednávání, neboť obvykle je třeba vyřešit a vyvážit i mnoho protichůdných požadavků.

Cílem analýzy a specifikace požadavků je stanovení služeb, které zákazník požaduje od systému a vymezení podmínek jeho vývoje a provozu.

3.1 Získání, analýza a definování požadavků

Je prvním krokem ke specifikaci požadavků. Je nutné, s využitím vhodných prostředků, zjistit, o jaký softwarový produkt má zákazník zájem, vymezit jeho funkcionalitu atd. V této fázi se zaměřujeme na uživatele, nikoli na to, jak lze docílit vytvoření projektu.

3.1.1 Transformace neformálních požadavků do strukturované podoby

Jednotlivé požadavky jsou obvykle od zákazníka získány v neformální podobě. Ty je nutné převést do částečně formálního, strukturovaného popisu. Tímto získáme jednoznačnější zadání a předejdeme nesrovnalostem v zadání.

3.1.2 Provedení studie vhodnost, identifikace a analýza rizik

Pokud známe požadavky zákazníka, měli bychom provést analýzu rizik. To znamená spočítat celkové náklady a zjistit, zda jsme za uvedenou cenu schopni vytvořit zákazníkem požadovaný výsledný softwarový produkt. Důležitý je také odhad doby, za kterou jsme schopni produkt předat. Na základě dostupných informací bychom měli být schopni říct, zda jsme schopni výsledný produkt vytvořit.

3.1.3 Plánování akceptačního testování

Jedná se o testy, které se uskuteční při předávání zákazníkovi. Mělo by být předem stanoveno, jaké výsledky budou považovány za úspěšné a které už nikoli. [2]

3.2 Určování požadavků

Určování požadavků je o společenských, komunikačních a manažerských dovednostech. Jedná se sice o první technickou fázi systémového vývoje, ale pokud nebude vytvořena důsledně, pak jsou další fáze mnohem obtížnější. Následkem nezachycení, vynechání, nesprávného výkladu zákazníkových požadavků může vést až ke krachu celého projektu.

3.2.1 Funkční a nefunkční požadavky

Systémové plánování určuje strategický směr pro organizaci. Účelem získávání požadavků je zajistit popis funkcí a jiných požadavků, které zákazník vyžaduje v implementovaném a vyvíjeném systému. Účelem získání požadavků je neformální popis funkčních a jiných požadavků, které zákazník vyžaduje v implementovaném a vyvíjeném systému.

Požadavky definují požadované služby a omezení, které musí systém splňovat. Tyto služby jsou tvořeny seznamem funkčních požadavků. Jednotlivé funkční požadavky mohou být seskupovány.

Funkční požadavky jsou získávány od zákazníka. Existuje mnoho technik, které mohou být použity, počínaje tradičním interview se zákazníkem a konče vytvořením softwarového prototypu, díky kterému získáme více požadavků.

Souhrn funkčních požadavků musí být podmíněn pečlivou analýzou k vyloučení duplicit a rozporů. To vede k vyhodnocení požadavků a k nové konzultaci se zákazníkem. Odsouhlasené funkční požadavky jsou modelovány pomocí grafických notací a popsány písemně.

Nefunkční požadavky se nechovají přirozeně. Jsou omezeny na vývoj a implementaci systému. Úroveň příslušnosti k těmto omezením určuje kvalitu softwaru. Nefunkční požadavky mohou být rozděleny na požadavky, které souvisí s:

- Použitelností
- Znovupoužitelností
- Spolehlivostí
- Výkonem
- Efektivností
- Podporovatelností
- Jinými vlastnostmi

[2]

3.2.1.1 Použitelnost

Definuje případy použití systému. Systém je použitelnější, čím více je uživatelsky přívětivější. Použitelnost je ovlivněna mnoha problémy, jako je dokumentace a nápověda, nezbytné zaškolení

pro efektivitu a účinnost použití, estetiku a konzistenci uživatelského rozhraní, zachycení chybových stavů, atp. Použitelnost je relativní pojem. Co je použitelné pro experta, může být pro začátečníka nepoužitelné a naopak.

3.2.1.2 Znovupoužitelnost

Znovupoužitelnost vyjadřuje, jak je obtížné znovu použít předchozí implementaci softwarových komponent v nově vyvíjeném systému. Znovupoužitelnost aplikujeme na uživatelské rozhraní, třídy, balíčky, frameworky, atp.

3.2.1.3 Spolehlivost

Označuje frekvenci a vážnost systémových selhání a jak snadno se systém obnoví po chybách. Spolehlivost je určena požadovanou dostupností systému v provozu (dostupnost znamená čas mezi chybami), přesnosti vytvořených výsledků, atp. Spolehlivý systém je takový systém, který je bezpečný za provozu (uživatelé se na něj mohou spolehnout).

3.2.1.4 Výkonnost

Je udávána očekávanou časovou odezvou systému, transakční výkonností, spotřebou systémových prostředků, možným počtem zároveň pracujících uživatelů, atp. Výkonnostní požadavky se mohou lišit pro různá prostředí (jiné nároky jsou např. pro bankovní systémy a jiné pro e-shopy).

3.2.1.5 Efektivnost

Souvisí s cenou a časem potřebným k realizaci projektu. Efektivita také zahrnuje ceny hardwaru, softwaru, lidí a jiných zdrojů. Efektivnější systémy použijí méně zdrojů na stejnou úlohu.

3.2.1.6 Podporovatelnost

Skládá se ze srozumitelnosti, udržovatelnosti a rozšiřitelnosti. Podporovatelnost je jednoduchost, s kterou můžeme systému porozumět, korigovat, dohlížet a rozšiřovat jej. Podporovatelnost se vyznačuje čistotou a jednoduchostí architektonického návrhu a přesnosti implementace podle návrhu. [2]

3.3 Získání požadavků

Získávání požadavků je jednou z nejdůležitějších fází vývoje softwaru. Získané požadavky definují, jakým směrem se bude ubírat vývoj a rozhoduje o konečné podobě výsledného softwaru. Na základě získaných požadavků se při vývoji zaměříme na podstatné detaily, které plynou z těchto požadavků, nepodstatné vypustíme. Tímto se zvýší efektivita práce a zkrátí se doba vývoje softwaru.

Systémový analytik zjišťuje systémové požadavky pomocí konzultací, které zapojí zákazníky a odborníky z oblasti problému. V některých případech systémový analytik má dostatečný přehled v dané problematice, a tak nevyžaduje pomoc zákazníků a odborníků, nebo jen omezeně.

Kdykoli při zachycení požadavků na softwarový systém pracujete s lidmi, snažíte se jejich prostřednictvím získat co nejpřesnější obraz nebo mapu jejich pracovního modelu. Podle Noama Chomského je tato mapa vytvořena třemi procesy:

- Vyřazením
- Deformací
- Zobecněním

Tento postup je naprosto nezbytný, protože nemáme k dispozici žádné poznávací zařízení, které by bylo schopno zachytit každou nuanci a podrobnost zkoumaného světa do nekonečně podrobné duševní mapy – musíme si tedy vybírat. Naše volba v nepřeberném množství možných informací musí projít následujícími výběrovými síty:

- Vyřazením, kdy je informace odfiltrována
- Deformací, kde je informace upravena souvisejícími mechanismy tvorby a halucinace
- Zobecněním, kdy dochází k tvorbě pravidel, víry a zásad týkajících se pravd a klamu

Tyto selektivní rysy jsou mechanismem utvářejícím přirozené jazyky. Je důležité o nich vědět – především tehdy, snažíte-li se o co nejpodrobnější zachycení požadavků a o jejich co nejdůkladnější analýzu. Můžeme totiž potřebovat jejich aktivní označení a vybídnutí k obnovení informace. [1]

3.3.1 Tradiční metody získání požadavků

Tradiční metody získávání požadavků zahrnují interview, dotazníky, pozorování, studium dokumentace. Tyto metody jsou jednoduché a cenově nenáročné. Efektivita tradičních metod je nepřímo úměrná ke stupni rizika projektu. Vysoký risk znamená, že systém je obtížné implementovat - dokonce vysoká úroveň požadavků není zcela zřetelná. V takových projektech jsou tradiční metody nevhodné.

3.3.1.1 Interview se zákazníky a doménovými experty

Interview jsou hlavní technikou k získávání faktů a shromažďování informací. Většina interview jsou vedeny se zákazníky. Interview se zákazníky většinou zajistí „use-case“ požadavky. Doménoví experti mohou být také dotazováni, když analytik nemá dostatek znalostí z dané problematiky.

Interview se zákazníky jsou komplexnější. Zákazníci mohou mít jen nejasné obrázky jejich požadavků. Mohou být neochotní nebo neschopní vyjádřit jejich požadavky ve srozumitelné podobě. Mohou také vznést požadavky, které převyšují rámec projektu, nebo jsou neimplementovatelné. V některých případech mohou být požadavky zákazníků protichůdné.

Existují dva typy interview: strukturované (formální) a nestrukturované (neformální). Strukturované interview je připraveno předem včetně otázek. Některé otázky mohou být otevřené, jiné mohou být uzavřené.

Strukturované Interview by mělo být doplněno nestrukturovaným interview. Pro nestrukturované interview jsou lepší neformální schůzky bez předem připravených otázek, nebo očekávaných cílů.

Jak strukturované, tak nestrukturované interview musí poskytnout podnět k diskuzi. Ta může proběhnout například emailem.

Je mnoho faktorů k uskutečnění úspěšného interview, ale asi nejdůležitější jsou tazatelovy komunikační a mezilidské schopnosti. Dokud tazatel pokládá otázky a udržuje kontrolu, je důležité pozorně naslouchat a mít strpení tak, že je dotazovaný uvolněný.

K udržení dobrého mezilidského vztahu a získání další zpětné vazby by měl být souhrn interview doručen dotazovanému během jednoho, nebo dvou dnů s požadavkem na jeho komentář.

3.3.1.2 Dotazníky

Dotazníky jsou účinnou cestou k získání informací od mnoha zákazníků. Dotazníky se obyčejně používají jako doplněk interview, ne místo nich. Výjimkou může být nízkorizikový projekt se srozumitelnými objekty.

Obecně, dotazníky mají menší vyjadřovací sílu, než interview. Dotazníky jsou pasivní, což může být výhodou i nevýhodou. Výhodou, protože odpovídající má čas si ujasnit odpovědi a může odpovídat anonymně. Nevýhodou může být, že odpovídající nemá potřebné znalosti z okruhu otázek.

Dotazník by měl být navrhnut tak, aby bylo jednoduché odpovídat. Měli bychom se vyhnout otevřeným dotazníkům, většina dotazníků bývá uzavřena. Uzavřené dotazníky mohou nabývat třech různých podob:

- **Vícenásobná volby** je dotazník, kde dotazovaný musí vyplnit jednu, nebo více odpovědí z předem připravených odpovědí.
- **Ohodnocení otázek** je dotazník, kde dotazovaný musí vyjádřit jeho názor o výrazu. Možné ohodnocení je pak plný souhlas, souhlas, neutrální, nesouhlas, úplný nesouhlas a nevím.
- **Ohodnocení otázek**, kde by odpovědi měly poskytovat ohodnocení s postupnými čísly, procentními hodnotami, nebo obdobnými metodami.

Dobře navržený jednoduchý dotazník bude podněcovat dotazované k brzkému odevzdání vyplněného dokumentu. Avšak, výsledky dotazníku nemohou být analytikem brány úplně v potaz, protože lidé, kteří neodpovídali, by mohli odpovědět v dotazníku jinak. [2]

3.3.1.3 Pozorování

Jsou situace, kde analytik obtížně získává kompletní informace pomocí interview a dotazníků. Zákazník může být neschopný sdělit informace efektivně a přesně, nebo může mít jen částečné

znalosti celého business procesu. V takových případech může být pozorování efektivní technikou hledání faktů. Pozorování může nabývat třech podob:

- **Pasivní pozorování**, kde analytik pozoruje pracovní business aktivity bez přerušení nebo zapojení. V některých případech mohou být použity videokamery pro hlubší a kontinuální pozorování.
- **Aktivní pozorování**, kde se analytik účastní aktivit. Stává se tak součásti týmu.
- **Výkladové pozorování**, kde uživatel vysvětluje jeho činnost při vykonávání jeho práce.

Reprezentativní vzorky pozorování by měly být získány v delších časových intervalech, v různých délkách a různých pracovních dobách.

Hlavním problémem pozorování jsou lidé, kteří mají sklon se při pozorování chovat nepřirozeným způsobem. Ve zvláštních případech mají tendenci pracovat podle stanovených pravidel a procedur. To pak zkreslí realitu skrytím některých úkonů, které mohou být důležité, ať už pozitivně, nebo negativně.

3.3.1.4 Studium dokumentů a softwarových systémů

Studium dokumentů a softwarových systému je neocenitelná technika pro sběr požadavků případu použití a znalostí požadavků. Technika je použita, ačkoli cílem jsou jen vybrané části systému.

Požadavky případu použití jsou vyhledávány v existujících organizačních dokumentech a systémových modelech/záznamech. Jedna z největší pomocí při návrhu požadavků případu použití jsou chybové záznamy stávajícího systému.

Studium organizačních dokumentů zahrnuje pracovní formuláře, pracovní procedury, popis práce, vnitřní pravidla, organizační struktura, komunikace mezi spolupracovníky, účetnictví, externí komunikace a stížnosti zákazníků.

Studium systémových záznamů a hlášení zahrnuje počítačové obrazovky a hlášení společně s příslušnou dokumentací (systémový manuál, uživatelskou dokumentaci, technickou dokumentaci a systémovou analýzu a návrh modelů).

Obor znalostí požadavků se vyhledává zkoumáním časopisů a knih z oboru projektu. Studium vlastností softwarových balíků, může také poskytnout bohaté informace z oboru. Z toho důvodu je návštěva knihoven a softwarového knihkupectví části procesu získávání požadavků. [2]

3.3.2 Moderní metody získávání požadavků

Moderní metody získávání požadavků zahrnují použití softwarových prototypů, brainstorming, spojení aplikačního vývoje (Joint Application Development JAD) a rychlý vývoj aplikací (rapid application development RAD). Tyto metody nabízí náhledy do požadavků, ale za vyšší cenu a úsilí. Avšak dlouhodobě mohou být velmi přínosné.

Moderní metody jsou typicky potřebné, když je riziko projektu vysoké. Faktorů pro vyšší riziko projektů je mnoho. Jsou včetně počátečních obtíží, dokumentovaných procedur, nestabilních požadavků, uživatelské neodbornosti a také nezkušených vývojářů.

3.3.2.1 Prototypování

Prototypování je nejčastěji používaná moderní metoda získávání požadavků. Softwarové prototypy jsou postaveny na vizualizaci systému, nebo jeho části, aby se zajistila zpětná vazba se zákazníky. Prototyp je demonstrační systém, „quick and dirty“ (rychlý a neohrabaný) pracovní model řešení, který poskytuje grafické uživatelské rozhraní (GUI) a simuluje chování systému pro různé uživatelské události. Informační obsah GUI obrazovek je natvrdo naprogramován.

Složitost moderního GUI dělá z prototypování nepostradatelnou část v softwarovém vývoji. Proveditelnost a užitečnost systému můžeme odhadnout pomocí prototypů ještě předtím, než se vytvoří reálná implementace.

Softwarový prototyp je velmi efektivní cestou získávání požadavků, které bychom jen obtížně získávali předchozími metodami.

Jsou dva typy prototypů:

- **Prototyp „zahození“ (throw-away)** je zahozen, když jsou požadavky kompletní. Má za cíl požadavky určující fázi životního cyklu. Typicky se soustředí na poslední známé požadavky.
- **Evoluční prototyp** se ponechá po získání požadavků a použije se k vytvoření konečného produktu. Evoluční prototyp má za cíl rychlé zhotovení konečného produktu. Typicky se soustředí na srozumitelné požadavky, takže první verze produktu může být zhotovena rychle, ale s nekompletní funkcionalitou. [2]

3.3.2.2 Brainstorming

Brainstorming je skupinová technika zaměřená na generování co nejvíce nápadů na dané téma. Je založena na skupinovém výkonu. Nosnou myšlenkou je předpoklad, že lidé ve skupině, na základě podnětů ostatních, vymyslí více, než by vymysleli jednotlivě. [3]

Obecně brainstorming není určen pro analýzu problému. Je pro vytváření nových nápadů, možných řešení a manažerských rozhodnutí. Analýzy a manažerská rozhodování pak následují a nezahrnují techniku brainstormingu.

Příklady otázek brainstormingu k získání požadavků:

1. Co bude systém umožňovat?
2. Jaké budou vstupní a výstupní data systému?
3. Jaké třídy jsou potřeba v business, nebo doménovém objektovém modelu?
4. Jaké otázky by měly být kladeny při interview nebo v dotazníku?
5. Které problémy ještě nebyly uvažovány?
6. Jaké jsou hlavní rizika projektu?

7. Jaké otázky by měly být položeny během následujícího brainstormingu?

3.3.2.3 Joint Application development

Joint application development (JAD) je brainstormingová technika. Cílem techniky JAD je aktivně zapojit uživatele do procesu analýzy a návrhu systému a zainteresovat je na jejich výsledku. Analýza požadavků, modelování, prototypování i návrh systému probíhá s využitím série organizovaných a usměrňovaných pracovních schůzek, kterých se účastní zástupci vedení firmy, uživatelé budoucího systému a IS/IT specialisté.

JAD pracovní schůzky vyžadují týmovou spolupráci všech účastníků. Trvají většinou několik dní. Schůzky mají charakter strukturovaných rozhovorů, jejichž smyslem je:

- Jaké jsou cíle zamýšleného systému, jejich priority a očekávané priority
- Od uživatelů zjistit, jaké jsou požadavky kladené na navrhovaný systém
- Zajistit shodný názor všech účastníků na specifikované cíle a požadavky
- Společně s uživateli budovat modely systému a prototypy
- Průběžně ověřovat, zda budovaný systém splňuje uživatelské požadavky tak, jak je systém navržen

[4]

3.3.2.4 Rapid application development

Rapid application development (RAD) je více než metoda pro získávání požadavků. Přistupuje k vývoji softwaru jako k celku. Jak už název napovídá, RAD znamená vytvoření systému rychle.

RAD kombinuje pět technik:

1. Evoluční prototypování
2. CASE nástroje s generováním kódu a zpětnou vazbou mezi návrhovým modelem a kódem
3. Odborníci s pokročilými nástroji - vývojový tým RAD. Nejlepší analytici, vývojáři a programátoři, kteří jsou ve firmě. Tým pracuje s těsným časově limitovaným plánem.
4. Interaktivní JAD – JAD porada, během které je autor nahrazen SWAT (Skilled Workers with Advanced Tools) týmem s CASE nástroji
5. Timeboxing – metoda projektového managementu, která využívá přesně daného času pro SWAT tým ke kompletaci projektu.

Problémy spojené s RAD:

- Nekonzistence GUI
- Spíše specializovaný, než obecně řešený
- Nedostatečná dokumentace

- Takový software je obtížné udržovat a rozšiřovat

3.3.3 Uspořádání a validace požadavků

Získané požadavky od zákazníků se mohou překrývat nebo být v konfliktu. Některé požadavky také mohou být dvojsmyslné nebo nerealistické. Jiné požadavky mohou být doposud nepokryté. Z těchto důvodů je potřeba, aby požadavky byly uspořádány a validovány před tím, než nalezneme cesty k dokumentování požadavků.

Uspořádání a validace požadavků nemůže být vypuštěna z procesu vypracování dokumentu získávání požadavků. Uspořádání požadavků je obvykle založeno na návrhu dokumentu. Seznamy požadavků v dokumentu návrhů jsou uspořádány a upravovány podle potřeby. Nesprávné požadavky jsou odstraněny. Nově získané požadavky jsou naopak přidány.

Validace požadavků vyžaduje úplnější verzi dokumentu požadavků se všemi jasně identifikovatelnými a klasifikovanými požadavky. Zákazníci si přečtou dokument a svolají formální hodnotící schůzku. Hodnocení je formou testování. [2]

4 Analýza

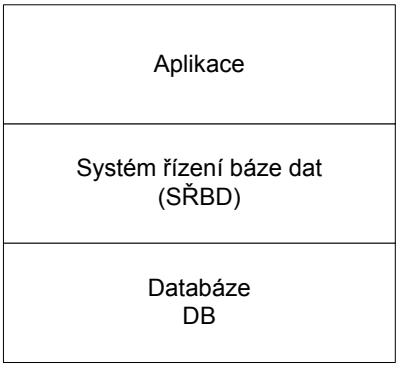
4.1 Databáze

Postupným vývojem od kartoték přes děrnoštítkové stroje po dnešní výkonné servery vznikly databázové systémy.

Databázi chápeme jako úložiště údajů, které jsou uloženy a zpracovávány nezávisle na aplikačním softwaru. Databáze zapouzdřuje jak data, tak relace mezi jednotlivými prvky a objekty v databázi.

Ve většině systému bývá databáze uložena v jednom souboru. Existují však i typy databází, v nichž je každá tabulka ve zvláštním souboru. Kromě problémů s omezenou funkcionalitou zde nastává problém s komunikací, což by sice nevadilo u desktopových systémů, ale u klientských systémů by se nepříjemně zatěžovaly komunikační kanály. Zátěž spočívá v tom, že desktopové databáze pracují s celým objemem dat. Tuto nevýhodu nevyloučilo ani vyčlenění funkčních enginů, které potom vystupovaly vůči jiným aplikacím v pozici serveru. Typickým příkladem je MS Jet Engine, což je výkonné jádro aplikace MS Access. Jestliže požadujeme informace z databáze klient-server, obvykle zašleme SQL dotaz a na základě výsledku dostaneme pouze požadovaná data. [17]

K databázi přistupují uživatelé pomocí klientských nebo konzolových aplikací. Komunikace probíhá principem požadavek-odpověď, kde požadavek představuje SQL dotaz a odpověď výslednou tabulku dat nebo informaci o provedené operaci. V širším smyslu jsou součásti databáze i softwarové prostředky, jež umožňují manipulaci s uloženými daty a přístup k nim. Tento systém se nazývá Systém řízení báze dat (SŘBD). Vrstva SŘBD usnadňuje práci uživateli. Ten nemusí mít znalosti o operacích, které se provádějí uvnitř databáze. Vrstvy SŘBD a DB pak tvoří databázový systém (DBS), viz obrázek 6.1.



Obrázek 4.1: Schéma databázového systému [17]

Databáze se dělí na pět základních typů:

- Síťová databáze
- Hierarchická databáze
- Relační databáze
- Objektová databáze
- Objektově relační databáze

Z těchto modelů databáze se prosadily zejména tři databázové modely, a to síťový, hierarchický a relační. Posledně jmenovaný je nejpropracovanější a také nejběžnější. V poslední době se však stále více prosazují databáze s objektovým modelem. Relačního modelu dat využívají například databázové systémy Microsoft SQL Server, Oracle, MySQL, PostgreSQL. Některé databáze umožňují ukládat a zpracovávat prostorová, multimediální data a jiná specializovaná data, příkladem je Oracle.

Víceuživatelský přístup k databázi předpokládá efektivní řízení přístupu několika uživatelů najednou nebo klientských aplikací. SŘBD tedy musí umět definovat přístupová práva pro jednotlivé uživatele nebo aplikace k databázovým objektům a také specifikovat rozsah jejich oprávnění.

Samotnou databázi můžeme ochránit před neočekávanými vlivy, jako je například výpadek elektrického proudu, zničení pevného disku, viry, aj. včasným zálohováním dat na externí média. Hardwarovým řešením může být kupříkladu zrcadlení pevného disku (RAID 1) nebo řešení pomocí RAID 5. Důležitým faktorem je také potřeba mít nainstalované nejnovější aktualizace operačního i databázového systému. Pro zvýšení bezpečnosti a zabránění úniku dat je pak dobré používat některý z antivirových softwaru v kombinaci s firewallem. V samotném databázovém systému je nutné přidělovat práva k dané části databáze jen těm uživatelům, kteří s ní opravdu potřebují pracovat a nenechávat jim práva k jiným částem, ke kterým nemají oprávnění. [17]

4.1.1 Relační databáze

Relační databáze je úložiště dat, které odpovídá relačnímu modelu. Základem relačních databází jsou databázové tabulky. Tabulky jsou tvořeny řádky a sloupci. V řádcích jsou uloženy záznamy entit, do sloupců ukládáme jednotlivé atributy entity. Jelikož by každý záznam v tabulce měl být jednoznačně identifikovatelný, je potřeba zajistit, aby jeden ze sloupců, nebo složení více sloupců, obsahovalo záznam, který jednoznačně identifikuje každý řádek tabulky. Takovéto sloupce se označují *primární klíč* (primary key). Ve sloupci označeném primárním klíčem nenalezneme dva totožné záznamy. Primární klíč je možné vložit současně na více sloupců tabulky, v tomto případě se mohou položky v jednotlivých záznamech opakovat. Nesmí však být stejné všechny položky primárních klíčů.

Relační databáze definuje, jakým způsobem je možné reprezentovat strukturu dat, způsob jejich ochrany a operace, které můžeme nad daty provádět.

4.1.2 SQL (Structured Query Language)

U relačních databází je základní výhodou relativně snadná modifikace a spojení tabulek a s nimi spojená možnost dotazů – dotazovací jazyky.

Jazyk SQL v sobě zahrnuje nástroje pro tvorbu databází (tabulek) – DDL a nástroje pro manipulaci s daty – DML. Ten obsahuje základní příkazy pro vkládání (*INSERT*), aktualizaci (*UPDATE*), mazání (*DELETE*) a projekci informací (*SELECT*). Jakým způsobem jsou tabulky ukládány, definuje jazyk SDL. SQL umožňuje také vytváření pohledů, což je virtuální tabulka složená z několika tabulek. Vytváření těchto pohledů obsahuje jazyk VDL .

Jazyk SQL byl postupně přijat jako standard různými výrobci databázových aplikací. Dotazy se zadávají na straně klienta a následně odesílají na server, který dotaz převezme, zpracuje a odešle výsledek zpět klientovi. Většina současných databázových systému z tohoto modelu vychází a v různé míře jej dodržují. [17]

4.1.3 T-SQL (Transact SQL)

Jedná se o systém, který se snaží o udržení konzistence dat v databázi. Jestliže by totiž došlo k výpadku systému během provádění operace, mohly by v databázi vzniknout chyby. T-SQL s sebou přináší zavedení řízení toků dat a jeden typ cyklu – while.

Systém provádí operace pomocí tzv. transakcí. Každá transakce obsahuje sadu příkazů SQL, jež mají být provedeny serverem. T-SQL před vykonáním každé transakce načte celou transakci do vyrovnávací paměti. Protože jsou příkazy uloženy v paměti, může systém vytvořit optimální prováděcí plán. Po nalezení chyby během vykonávání dotazu je transakce ukončena nebo je možné ji celou odvolat. Při výpadku během provádění transakce se vrátí databáze do stavu před vykonáním přerušené transakce. Cenou za použití T-SQL je však snížení výkonu systému zapříčiněnou archivací transakcí a jejich optimalizací.

4.1.4 Uložené procedury

Uložené procedury rozšiřují standard SQL a využívají výhod T-SQL. Pro vyšší flexibilitu při psaní kódu je zavedena podmínková logika. Procedury jsou uloženy přímo v databázi, čímž se snižuje objem přenesených dat a tím také čas vykonání celé operace. Některé dotazy se stále opakují, takže je mnohem efektivnější volat jeden příkaz, který interpretuje celý dotaz. SQL server uložené procedury předkompiluje a tím se zajistí další zkrácení doby mezi zasláním dotazu a obdržením odpovědi. Vývojáři klientských aplikací tak jsou oproštěni od komplexních znalostí mnohdy rozsáhlých databází a vystačí si s názvy, vstupními parametry a výstupními hodnotami uložených procedur.

Předkompilování neznamená převedení na binární soubor, na spustitelný soubor, ani vytvoření samostatného souboru obsahujícího tuto uloženou proceduru. Zkompilováním je myšleno to, že server vytvoří a spravuje tzv. prováděcí plán. Díky tomu je optimalizátor, který zkoumá zatížení procesoru, paměti a disku, obvykle schopen najít nejrychlejší způsob, jakým proceduru provést. Prováděcí plán je vybírán z několika variant a může se v průběhu času měnit. Výběr nesprávného plánu může mít za následek zpomalení dotazu i o několik desítek sekund v případě spojování tabulek s velkým počtem řádků.

Zavedením uložených procedur vnášíme do databáze riziko zneužití zvenčí. Uložené procedury mohou provádět jakékoliv administrativní úkony, měnit přístupová práva nebo také přistupovat do registrů. Proto je vhodné se vyvarovat neznámým nezdokumentovaným předkompilovaným procedurám.

Existují dvě kategorie procedur: normální a kompilované. Kompilované jsou uloženy v DLL knihovnách, a jelikož tyto knihovny nemusí být zdokumentovány, mohou ukrývat potencionálně nebezpečný kód. U normálních uložených procedur má uživatel možnost prohlédnout zdrojový kód a zjistit, co daná procedura provádí, jaké parametry používá a jaké naopak vrací.

Server má k dispozici množství systémových uložených procedur, jejichž názvy začínají písmeny „*xp_*“, nebo „*sp_*“. Jako velmi nebezpečná je označována procedura „*xp_cmdshell*“, která umožňuje spustit kterýkoli příkaz operačního systému pod účtem pod kterým je spuštěn SQL server.

Speciálními typy uložených procedur jsou databázové triggery (nebo také aktivační procedury). Aktivační procedury nejsou volány aplikací či uživatelem, ale změnou dat v databázových tabulkách. Můžou být vyvolány přidáním, odebráním nebo aktualizaci dat. Při použití tohoto typu uložené procedury snižujeme výkon databázového systému, jelikož se při každé aktualizaci dat zjišťuje, zda se k dané tabulce neváže nějaký trigger.

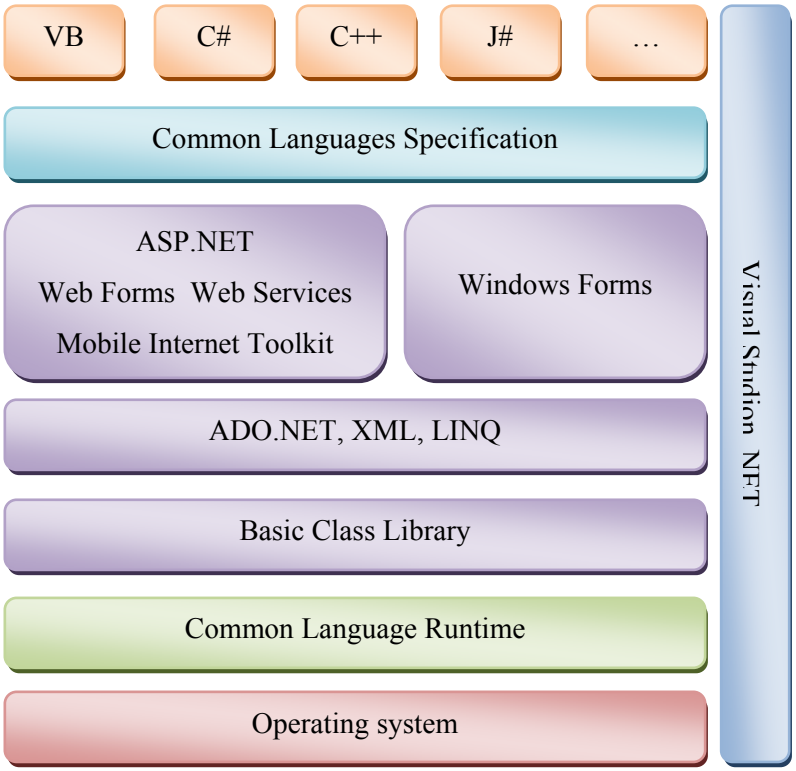
4.2 Microsoft Visual Studio .NET

Jedná se o balení vývojářských nástrojů a komponent, které umožňují snadnější a rychlejší vývoj softwaru.
Toto balení obsahuje několik druhů programovacích jazyků, kterými jsou:

- Microsoft Visual Basic
- Microsoft Visual C#
- Microsoft Visual C++
- Microsoft Visual J#

Ve všech těchto jazycích je možné vytvářet programy od konzolových aplikací, přes rozsáhlé informační systémy, až po software pro mobilní zařízení.

Při psaní kódu je velmi využívána funkce *IntelliSense®*. *IntelliSense®* nabízející rychlou nápovědu metod, vlastností, parametrů a povolených hodnot pro definovaný objekt. Při rozbalení nabídky povolených hodnot (objektů) stačí jednoduše kurzorem vybrat prvek ze seznamu a vložit jej tak do programového kódu.



Obrázek 4.2: Diagram hierarchie .NET Frameworku [15]

Visual Studio je určeno pro vývoj aplikací na platformě .NET Framework. .NET Framework využívá pro komunikaci s databází několik tříd, které plně využívá Visual Studio .NET. Pro spojení a udržování spojení s databází je zde použita třída *SqlConnection*. K úspěšnému připojení je nutné vyplnit *Connection String*. Další z nich je *SqlCommand*, pomocí kterého je možné nastavovat parametry pro SQL příkazy. *SqlDataReader* umožňuje jednosměrnou práci s daty databáze. Je možné načítat záznamy z databáze po jednotlivých řádcích, nelze však data ukládat, ani je nijak měnit. K obousměrné komunikaci je zde *SqlDataAdapter*. Objekty typu *DataSet* v sobě zahrnují datové tabulky a operace s nimi.

Novinkou v .NET 3.5 je LINQ to SQL, který přistupuje k databázím jiným způsobem, než je předchozí případ.

4.3 Jazyk C#

Tento programovací jazyk patří mezi nejmladší programovací jazyky vůbec. Jedná se o objektově orientovaný jazyk, jež je odvozen z jazyka C/C++ a Javy. Zavedením tohoto jazyka byly eliminovány některé potencionálně nebezpečné rysy jazyka C/C++, kterými jsou ukazatele či nepřímá adresace proměnných. Pro správu paměti přibyl garbage collection (správa paměti). C# je postaven nad platformou .NET, což nám umožňuje využit jeho bohatou knihovnu funkcí.

.NET Common Language Runtime je prostředí založené na komponentách a je tedy předurčené ke snadnému vytváření a práci s komponentami. Jedná se proto o jazyk, kde se všechny objekty programují jako komponenty. K nim lze přiřazovat atributy, které mohou předávat informace o těchto komponentách ostatním součástem systému.

Automatické přidělování paměti ulehčuje mnohdy komplikovanou práci programátora při alokaci a uvolňování paměti a případně následnému hledání paměťových úniků. Jazyk je typově bezpečný a značně tak zamezuje práci s neinicializovanými proměnnými a používání nebezpečných přetypování.

Již existující objekty typu COM lze použít jako objekty typu .NET. Prostředí .NET CLR zajistí, že objekty .NET se budou chovat k ostatním objektům COM jako objekty COM. Je možné také volat kód vytvořený jazykem C a následně umístěný v knihovně DLL. [15]

4.3.1 Kolekce .NET 3.5

LINQ dokáže zpracovat kolekce, které implementují generické rozhraní IEnumerable<T> a poli.
Vyhledaní a setřídění prvků pole může vypadat například takto:

```
int[] posloupnost = new int[6] { 3, 7, 2, 5, 4, 6 };
//vyhledaní lichých prvků
IEnumerable<int> vysl =
      posloupnost.Where(o => o % 2 != 0).OrderBy(o => o);
```

V předchozím příkladu bylo použito tzv. Extensions methods, které rozšiřují veškeré implementace rozhraní IEnumerable<T> o řadu nových metod pro práci s daty.

V extensit metodách je často povinný parametr Func<T>. Jedná se o delegáta, který je obecný. Vstupní data jsou pak předávána formou generických argumentů a na základě nich je také určena návratová hodnota. Instance těchto delegátů může být tvořena pomocí lambda výrazů, tím se zkrátí zápis anonymní metody.

4.3.2 Nová klíčová slova C# 3.0

Dotazování v .NET Frameworku 3.5 je umožněno i jiným způsobem, než pomocí Extension Methods. Byly zavedeny nová klíčová slova (query keywords) pro skládání dotazů. Díky nim lze napsat předchozí příklad např. takto:

```
int[] posloupnost = new int[6] { 3, 7, 2, 5, 4, 6};
//vyhledani lichých prvků
IEnumerable<int> vysl = from cislo in posloupnost
                        where cislo % 2 != 0
                        orderby cislo
                        select cislo;
```

Tyto dotazy připomínají dotaz v jazyce SQL s tím rozdílem, že *select* je na konci dotazu. Klíčové slovo *from* je na začátku dotazu, to umožňuje efektivně využívat funkci Visual Studia IntelliSense (interaktivní zobrazování dostupným příkazů uživateli).

Při přeložení stejných dotazů napsaných v query keywords a pomocí Extension methods a prohlédnutím IL kódu zjistíme, že oba kódy jsou stejné. Query keywords jsou převedeny na Extension methods. Je jen na programátorovi, aby si vybral, která metoda se mu více zamlouvá, na výsledek to nemá žádný vliv.
[9][10]

4.4 LINQ

Jedním z největších problémů většiny vývojových prostředí je přístup k různým datovým zdrojům. Práce s databázemi a manipulace s XML zdroji jsou často označovány jako nejlepší, ale zároveň také velmi problematické datové zdroje. Proto se hledal způsob, jak zefektivnit, zpřehlednit a sjednotit zpracování dat z různých typů datových zdrojů.

Projekt LINQ (LAnguage Integrated Query) je krycí jméno pro množinu .NET Frameworku jako jsou jazykově integrované dotazy, množiny a transformační operace. Rozšiřuje C# a Visual Basic nativní jazykovou syntaxí pro dotazy a poskytuje třídy knihoven pro lepší zpracování těchto rozšíření.

Od počátku .NETu bylo několik způsobů, jak pracovat s relačními a hierarchickými daty. Nejpoužívanějším pro zpracování relačních dat je ADO.NET, který umožňoval zpracovávat data jak v online, tak v offline režimu. Ke zpracování hierarchických dat ve formátu XML jsou zde k dispozici jmenném prostoru *System.XML*. Zpracovávat tyto data lze pomocí DOMu nebo sekvenčně. LINQ je však nová technologie, která pracuje s daty odlišným způsobem.

LINQ přináší do programovacích jazyků .NET podporu dotazování. Z integrace do programovacích jazyků plyne odhalení mnoha chyb již v době kompilace, takže chyby se neprojeví až za běhu aplikace.

Pomocí LINQ lze zpracovávat téměř jakékoli kolekce dat, protože jeho architektura je navržena tak, že je možné tvořit její implementace pro jednotlivé datové zdroje. Je zde jistá podobnost s ADO.NET, kde lze zpracovávat různé typy databází. LINQ je však navrhnut mnohem abstraktněji a lze tak zpracovávat i nedatabázová data. Této vlastnosti je dosaženo především pomocí výrazových stromů (ExpressionTrees), který umožňuje pracovat s kódem jako s daty. LINQ pak může být přeložen adekvátním providerem pro daný zdroj. Díky tomu je jedno, jestli je zdroj relační databáze, nebo pole.

Samotný .NET framework 3.5 obsahuje několik implementací LINQ. Jedná se o následující implementace:

- **LINQ to Objects** – pro standardní kolekce nacházející se v paměti (např. pole, seznamy)
- **LINQ to SQL** – pro MS SQL 2000 a vyšší
- **LINQ to XML** – pro práci s XML daty
- **LINQ to Dataset** – pro práci s ADO.NET datasety

[10]

4.4.1 LINQ to SQL

LINQ to SQL je aplikační programové rozhraní (API) pro práci s SQL databázemi. V současných objektových programovacích jazycích nastává problém, jak komunikovat s relačními databázemi. Při psaní aplikací modelujeme části reálného světa reprezentované třídami. Potřebujeme zajistit perzistenci těchto objektů, aby při restartu aplikace byly zachovány všechny její objekty. Avšak většina databázových systémů je relační, takže ukládáme záznamy tabulek a ne objekty.

Další problém může nastat v tom, že databázový server používá jiné datové typy, než programovací jazyk, v kterém je aplikace napsána. Programátor tak musí hlídat datové typy, které používá databáze a které může použít v programovacím jazyce. Při nesprávně zvoleném typu může nastat záměna hodnoty, nebo v horším případě pád aplikace.

LINQ to SQL je možné používat jen pro SQL Server 2000 a novější a SQL Express. Pro práci s jinými databázemi lze použít SQL to Dataset.

Dodatkem k LINQ to SQL je vrácení jednoduchých polí, neentitních tříd, nebo anonymních tříd. LINQ to SQL také zajišťuje sledování změn v datech a jejich aktualizaci v databázi s detekcí souběžné kolize a jejím optimistickým řešením s transakční integritou. [10]

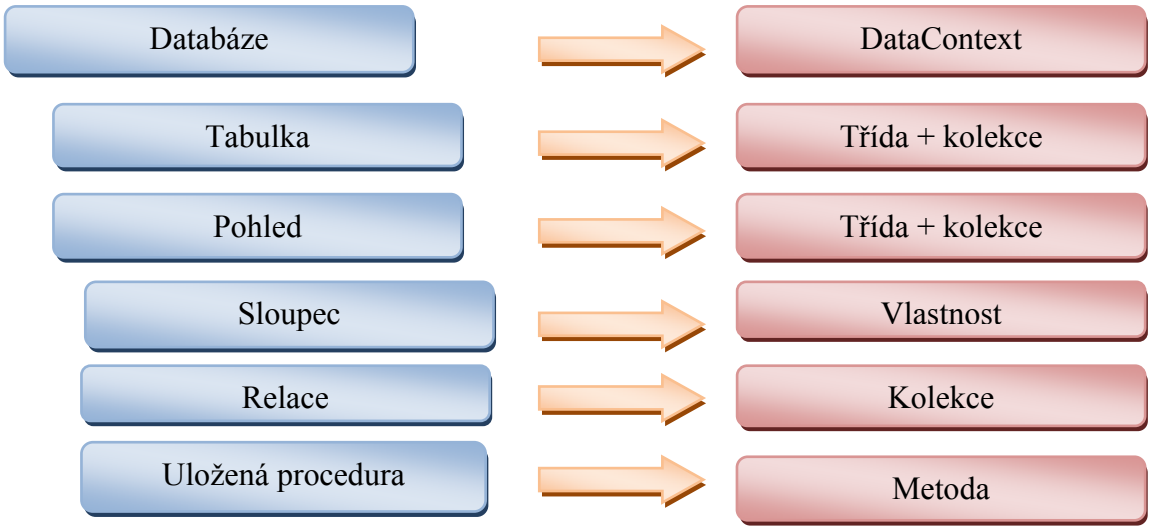
4.4.2 Mapování LINQ to SQL

Při použití LINQ to SQL se typicky používá třída odvozená z třídy DataContext. Jméno třídy je obvykle stejné jako jméno databáze. Odvozená třída od DataContext obsahuje veřejné proměnné typu *Table<T>* pro každou databázovou tabulku, kde *T* je typ entitní třídy, která je instancí pro každý získaný záznam z příslušné databázové tabulky.

Datový typ Table<T> je specializovaná kolekce. LINQ to SQL zajišťuje použití entitních tříd, kde je obvykle každá entitní třída vázána na jednoduchou databázovou tabulku. Vlastnosti (properties) entitní třídy jsou vázány k sloupcům databázové tabulky. Základem LINQ to SQL je mapování entitních tříd do databázových tabulek a entitních vlastností do databázových sloupců.

Asociace je výraz užívaný k určení primárního a cizího klíče mezi dvěmi entitními třídami. U relací *1:N* je výsledek asociace v rodičovské třídě. Třída obsahující primární klíč obsahuje kolekci tříd potomků, tyto třídy obsahují cizí klíč. Tyto kolekce jsou uloženy a označeny jako soukromé proměnné typu *EntitySet<T>*, kde *T* je typ potomka entitní třídy. Na opačném konci relace, která obsahuje cizí klíč je odkaz na rodičovskou třídu, pokud se jedná o relaci *1:N*. Tato reference je uložena jako soukromá proměnná typu *EntityRef<T>*, kde *T* je typ rodičovské třídy.

Uložené procedury a uživatelsky definované funkce jsou v LINQ to SQL reprezentovány jako metody. [8]



Obrázek 4.4: Mapování databáze do objektů

4.5 Překlad LINQ na SQL

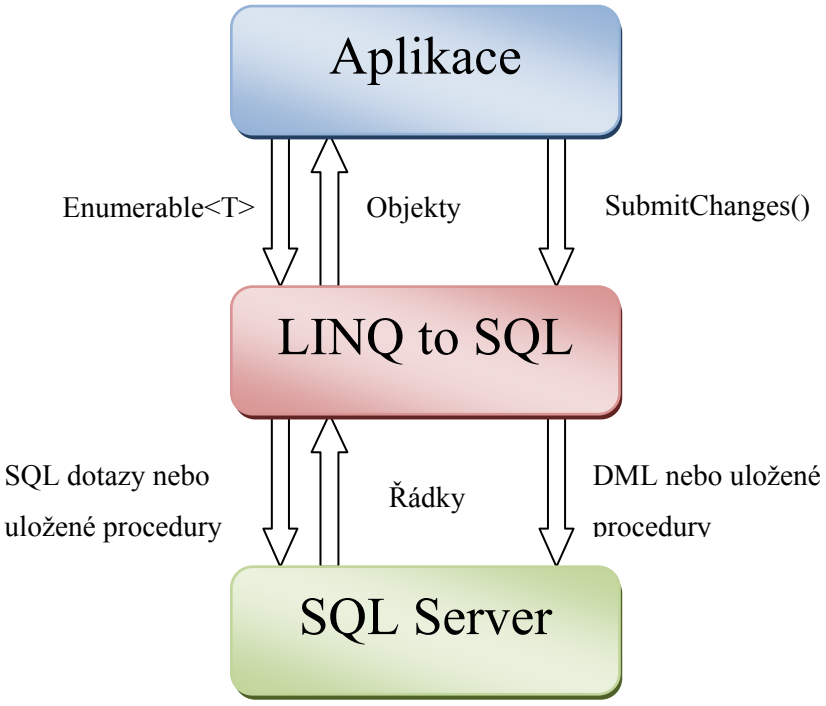
LINQ to SQL dotazy vrací sekvence typu IQueryable<T>, které nejsou kompilované do mezikódu, jako obyčejné LINQ dotazy. Místo toho jsou převedeny do výrazových stromů, které jim umožní vyhodnocení jako jednoduchou jednotku a přeložit ji do příslušných optimálních SQL dotazů. Na rozdíl od LINQ dotazů, které jsou vykonány v lokální paměti, LINQ to SQL jsou přeloženy na SQL volání a vykonány v databázi. [8]

Obrázek 4.3: Mapování LINQ do SQL

```
Customer cust = (from c in db.Customers
where c.Name == "Novák"
orderby c.Name
select c);
```

Dotaz, který volá LINQ to SQL do database:

```
SELECT *
FROM Customers
WHERE Name='Novák'
ORDERBY Name
```



Obrázek 4.5: LINQ to SQL

4.5.1 Mapování databáze do objektů

Při použití LINQ to SQL se typicky používá třída odvozená z třídy DataContext. Jméno třídy je obvykle stejné jako jméno databáze. Odvozená třída od DataContext obsahuje veřejné proměnné typu *Table<T>* pro každou databázovou tabulku, kde *T* je typ entitní třídy, která je instancí pro každý získaný záznam z příslušné databázové tabulky.

Datový typ Table<T> je specializovaná kolekce. LINQ to SQL zajišťuje použití entitních tříd, kde je obvykle každá entitní třída vázána na jednoduchou databázovou tabulku. Vlastnosti (properties) entitní třídy jsou vázány k sloupcům databázové tabulky. Základem LINQ to SQL je mapování entitních tříd do databázových tabulek a entitních vlastností do databázových sloupců.

Asociace je výraz užívaný k určení primárního a cizího klíče mezi dvěmi entitními třídami. U relací *1:N* je výsledek asociace v rodičovské třídě. Třída obsahující primární klíč obsahuje kolekci tříd potomků, tyto třídy obsahují cizí klíč. Tyto kolekce jsou uloženy a označeny jako soukromé proměnné typu *EntitySet<T>*, kde *T* je typ potomka entitní třídy. Na opačném konci relace, která

obsahuje cizí klíč, obsahuje odkaz na rodičovskou třídu, pokud se jedná o relaci *1:N*. Tato reference je uložena jako soukromá proměnná typu *EntityRef*<*T*>, kde *T* je typ rodičovské třídy.

Uložené procedury a uživatelsky definované funkce jsou v LINQ to SQL reprezentovány jako metody. [8][10]

4.6 Informační systémy

Informační systém je systém sběru, uchovávání, analýzy a prezentace dat určených pro poskytování informací mnoha uživatelům různých profesí.

Pro informační systém platí několik faktů:

- Může, ale nemusí být podporován počítačem, přičemž při návrhu IS zkoumáme optimální kombinaci automatizovaných a neautomatizovaných činností.
- Musí disponovat prostředky sběru, kontroly a uchování dat.
- Jsou vyjasněné vztahy mezi informacemi a daty i v rámci jednoho zaměření informačního systému. Informace jsou jen ta data, která dokážeme využít, přiřadit jim význam či smysl. Při návrhu IS je nutno umožnit získávání odlišných informací pro různé uživatele.
- Informační systém ovlivňuje pracovní procesy i organizační struktura podniků.
- Informační systém je vždy společným dílem dodavatele a zákazníka.

Další důležitou otázkou je, jak lze informační systém získat. Máme v podstatě dvě základní možnosti. Můžeme informační systém vyvíjet od začátku přesně na míru zadaným požadavkům, avšak s vidinou velkých nákladů, dlouhé doby vývoje a chyb prvních pokusů, nebo můžeme informační systém koupit a přizpůsobit našim potřebám. Druhý způsob se nazývá *customizace* a stejně jako vše ostatní má zcela pochopitelně své klady i zápory.

Uživatelé dnes IS obvykle nevyvíjejí. Vývoj naprosté většiny informačních systémů i customizace jsou totiž obvykle prováděny specializovanými firmami a nikoli přímo uživatelem.

S návrhem IS úzce souvisí obor zvaný Softwarové inženýrství. Zabývá se zásadami vývoje softwaru, jeho instalací a udržováním v provozu. Především se však jedná o systematický přístup k analýze návrhu, zavádění a údržbě softwaru.

Proces návrhu systému:

1. **Definice parametrů systému:** Zde si pokládáme otázku, co vlastně od systému požadujeme.
2. **Definice pracovních procesů:** Musíme si uvědomit, jaké úkony bude uživatel v systému provádět a co od něj bude očekávat.
3. **Seskupení myšlenkového datového modelu:** Smyslem sestavení myšlenkového datového modelu není jenom definice datových struktur pro uložení dat, ale využití dat v celém systému.

4. **Příprava databázového schématu:** Převod datového modelu do konkrétních fyzických pojmů.
5. **Návrh uživatelského rozhraní:** Jedná se o nadstavbu celého systému. Z hlediska uživatele je však tím prvním a zároveň posledním co ho zajímá.

První čtyři body spadají do analýzy systému a hrají tedy velmi důležitou roli v návrhu celého informačního systému. Pokud v této části uděláme závažnější chybu, může to mít katastrofální důsledky a celý systém můžeme budovat znovu od začátku. Realizace datového modelu se provádí nejčastěji pomocí ER diagramů, DFD diagramů, diagramů případu použití nebo univerzálního jazyka UML.

Dalším důležitým aspektem je výběr vhodného programovacího jazyka a databázového systému. Je důležité zvážit, který systém je pro vývoj systému nejvýhodnější. Podstatnou roli hraje rychlost zpracování v daném vývojovém prostředí, další, neméně důležitou, pak také cena produktu. Dalšími kritérii pro výběr mohou být také nutnost zálohování, replikace, rozložení zátěže na více serverů, aj. Pro vývoj webového informačního systému se nejčastěji využívá kombinace PHP a MySQL, nebo kombinace ASP a MSSQL. Klientské informační systémy nalezneme nejčastěji vytvořené v jazycích C++ či Javě, poslední dobou se rozmáhá i C#, případně Visual Basic.

Následuje implementace systému, kde volíme, jak se bude systém vytvářet. Asi nejvhodnější pro vytváření rozsáhlejší systému je zvolení tzv. modulárního přístupu. Každý modul (komponenta) může pracovat nezávisle na systému a také ji lze otestovat před připojením do systému.

Po implementaci následuje fáze testování a zavádění systému do provozu. Chyby systému je nutno hlásit tvůrci. Je vhodné při závadě nebo chybě zapsat oznámení do logovacího souboru, případně přímo odeslat tvůrci programu. [16]

4.7 Microsoft .NET

Pod termínem “Microsoft .NET” si lze představit poměrně značnou snahu společnosti Microsoft, jak již ostatně sám název napovídá, o změnu v tradičním vývoji softwaru a zapojení Internetu do určité softwarové platformy orientované na služby. To vše s pomocí partnerů v celém počítačovém světě. V této souvislosti se často setkáváme s pojmem XML, konkrétněji však se slovním spojením „webová služba XML“. Taková služba není nic jiného, než nějaká aplikace běžící na webovém serveru a vystavující *webové metody*, odborněji volatelné funkce API, klientům na Internetu. XML se ve zmíněném označení vyskytuje z velice prostého důvodu. Je to totiž jazyk, který využívají webové služby a jejich klienti k výměně dat. Jelikož se XML stává již v podstatě standardem pro popis různých typů informací, dá se zcela logicky předpokládat, že se budou webové služby XML velice rychle rozšiřovat. To bude mít za následek fakt, že se Internet stane softwarovou platformou s API prostředím mnohem bohatším, než může nabídnout jakýkoli operační systém. Zatímco se tak dnešní

aplikace spoléhají především právě na operační systém, další vize je, aby aplikace moderní, samozřejmě stále za podpory operačního systému, využívaly daleko více webové služby.

Je potřeba poznamenat, že webové služby nejsou původním dílem společnosti Microsoft. Ačkoli lze tuto společnost považovat za přední v oblasti vývoje a definice norem, sama žádné ze standardů, z nichž webové služby vychází, nevlastní. Využívají se naopak tzv. otevřené standardy, jakými jsou HTTP, XML a SOAP. Poněvadž se tedy jedná o průmyslové normy, a nikoli standardy společnosti Microsoft, webové služby se již pochopitelně na Internetu rozvíjejí. Nepracují sice většinou na systému Windows, avšak internetové služby jsou založeny na interoperabilitě. Je tedy relativně snadné vytvořit například klienta služby, jež běží na Windows a volá metody webové služby pracující pod Linuxem.

Jelikož je zřejmé, že lze psát webové služby bez pomoci společnosti Microsoft, nabízí se vcelku logicky otázka, co vlastně Microsoft .NET přináší. Odpověď je však snad až příliš prostá. Diskutovaná iniciativa má totiž pouze programátorům usnadnit jejich práci při vytváření a zpřístupňování služeb na Internetu, což umožňuje se mnohem lépe soustředit na obchodní logiku a vlastní služby. S trochou nadsázky lze i říci, že díky možnosti nezabývat se komunikačními protokoly a infrastrukturou zvládne vytváření webových služeb prakticky každý.

4.7.1 Microsoft .NET Framework

Nedílnou a zřejmě klíčovou součástí Microsoft .NET je platforma .NET Framework. Její potenciál, ačkoliv bylo doposud debatováno výhradně o webových službách XML, je poměrně široký. Platforma .NET Framework je tedy určena vývoj následujících typu aplikací:

- webové služby založené na XML
- webové formuláře
- aplikace s grafickým uživatelským rozhraním (GUI) Win32
- aplikace s konzolovým uživatelským rozhraním (CUI) Win32
- služby Windows (řízené správcem Service Control Manager)
- pomocné programy a samostatné komponenty

V souvislosti s webovými formuláři je ještě rovněž zřejmě vhodné zmínit model ASP.NET, jež je součástí platformy .NET Framework. Podobně jako aktivní serverové stránky – ASP dokázaly ve webovém programování devadesátých let minulého století udělat v podstatě revoluci, konkrétně díky svému snadno použitelnému modelu dynamického generování HTML na webových serverech, posunuje ASP.NET webové programování kupředu zavedením opakovaně použitelných serverových ovládacích prvků. Ty omezují HTML na prohlížečové klienty a spouštějí události, jež lze zpracovávat skripty na straně serveru. ASP.NET tak poskytuje vcelku lákavý nový způsob psaní webových aplikací, který se nepodobá ničemu zatím existujícímu.

Platforma .NET Framework sestává ze dvou základních částí. Těmi jsou společný jazykový běhový modul – CLR a knihovna tříd rámce .NET – FCL.

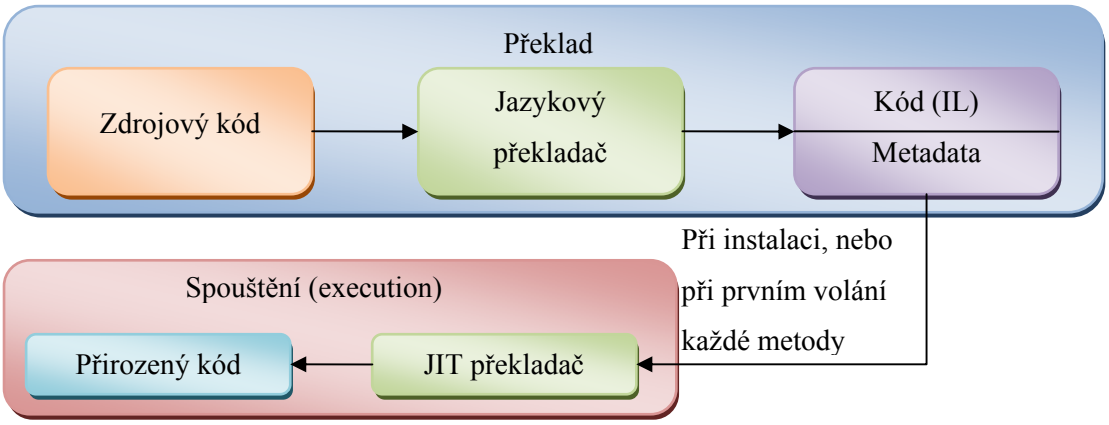
4.7.1.1 Společný jazykový běhový modul

V případě, že bychom si dokázali rámec .NET Framework představit jako živou bytost, byl by společný jazykový běhový modul jeho srdcem. Totiž každý námi zapsaný bajt kódu pro .NET Framework běží buď v CLR (Common Language Library), nebo mu alespoň CLR umožní běžet mimo tento modul. Bez akce CLR se zkrátka nic nestane.

Společný jazykový běhový modul se nachází nad operačním systémem a poskytuje ideální virtuální prostředí pro hostování řízených aplikací. V okamžiku, kdy otevřeme řízený vykonatelný soubor, nahraje CLR modul obsahující příslušný spustitelný soubor a vykoná kód v něm obsažený. Kódy, jež jsou zacíleny na CLR, se označují jako řízené (někdy též spravované) kódy a skládají se z instrukcí zapsaných v pseudo-strojovém kódu. Ten označujeme jako *společný zprostředkovací jazyk* neboli CIL, jehož instrukce se na požádání (JIT) za běhu zkompilují do nativního strojového kódu. Ve většině případů se určitá metoda zkompiluje pomocí techniky JIT pouze jednou, konkrétně při svém prvním volání, přičemž se následně uloží do paměti, aby tuto metodu bylo příště možno vykonat bez zpoždění. Kód, který se nikdy nezavolá, se ani nezkompiluje. Ačkoliv kompilace JIT prokazatelně snižuje výkonnost, jsou její negativní efekty zmírněny skutečností, že daná metoda se kompiluje pouze jednou, a také tím, že ze strany společnosti Microsoft bylo věnováno maximální úsilí k tomu, aby byl kompilátor JIT rychlý a výkonný, jak jen to je možné.

Výhod, jež nabízí provozování kódu v řízeném prostředí CLR je mnoho. Pokud kompilátor JIT převádí instrukce CIL na nativní kód, je k tomu využíván proces ověřování kódu, který zajišťuje typovou bezpečnost daného kódu. Je tak v podstatě nemožné vykonat nějakou instrukci přistupující k paměti, k níž nemá CLR oprávnění. V řízené aplikaci tedy v podstatě nehrozí problémy se ztracenými ukazateli, jelikož CLR v případě použití tohoto ukazatele, tedy před jeho použitím, vyvolá výjimku. Nelze ani zavolat metodu s nekorektně zapsaným rámcem zásobníku, protože v CLR k tomu zkrátka nemůže dojít. Vyjma eliminování těch nejběžnějších chyb, které mohou nepříznivě ovlivňovat aplikační programy, existuje také zabezpečení ověřováním kódu, jež výrazně znesnadňuje vytváření zlomyslného kódu úmyslně škodícímu hostitelským systémům. Zabezpečení ověřováním kódu je rovněž zásadní technologií, která dává CLR možnost hostit více aplikací v jediném procesu. Tento trik spočívá v rozdělení procesu na virtualizované oddíly označované jako *aplikační domény*; systém Windows vzájemně odděluje aplikace tím způsobem, že je hostí v oddělených procesech. Nevhodným vedlejším důsledkem modelu samostatného procesu pro každou aplikaci je však vyšší spotřeba paměti. Výkonné využití paměti sice není příliš důležité na samostatných systémech sloužících jednomu uživateli, ovšem je absolutně klíčové na serverech, jež jsou nastaveny na nárazové zpracování dotazů tisíců uživatelů. V jistých případech, například aplikace ASP.NET a webové služby, nespouští CLR proces pro každou aplikaci, namísto toho dochází k tomu, že spouští jeden proces nebo několik málo procesů a hostí jednotlivé aplikace ve zmíněných aplikačních doménách.

Jiné pozitivum provozování v řízeném prostředí pramení z toho, že prostředky alokované řízeným kódem se automaticky uvolňují z paměti. Tedy jinak řečeno, my sice paměť alokujeme, avšak její uvolnění již za nás dělá systém. CLR v sobě zahrnuje velice užitečný nástroj uvolňování paměti, který sleduje odkazy na objekty vytvářené našim kódem, přičemž tyto objekty, v případě potřeby obsazené paměti jinde, ničí. Díky tomuto nástroji nezpůsobují aplikace, skládající se výhradně z řízeného kódu, žádné úniky. Ba naopak, uvolňování paměti dokonce zvyšuje výkonnost, neboť algoritmus alokování paměti používaný v CLR je rychlý – pro srovnání mnohem rychlejší než odpovídající rutiny alokování paměti v runtimovém modelu jazyka C. Nevýhodou ale je, že v momentu uvolňování paměti se vše ostatní v daném procesu na okamžik zastaví. Tento vliv na výkonnost se však naštěstí radikálně snižuje tím, že k uvolňování paměti dochází relativně zřídka.



Obrázek 4.6: Diagram popisující překlad kódu .NET Frameworku [14]

4.7.1.2 Knihovna tříd rámce .NET

Druhou zásadní komponentou .NET Framework je knihovna FCL (Framework Class Library) obsahující tisíce definic typů, přičemž každý z těchto typů nabízí určité funkce. Jedná se tedy o třídy, struktury, rozhraní, výčty a delegáty, které jsou integrální součástí .NET Framework. Některé třídy obsahují i více než 100 metod, vlastností a dalších členů, takže se FCL nelze naučit úplně snadno. Špatnou zprávou je i skutečnost, že je to podobné jako se seznamovat s úplně novým operačním systémem. Tato zpráva je však vyvážena zprávou dobrou, neboť každý jazyk používá stejné API. To v praxi znamená, že pokud se my nebo společnost nás zaměstnávající rozhodne přejít například z Visual Basicu na C++ či naopak, naše předchozí píle se rozhodně neztratí.

Z toho důvodu, aby bylo učení a používání FCL zvládnutelnější, rozdělila společnost Microsoft knihovnu tříd do hierarchických jmenných prostorů. FCL tak tedy má přes 100 jmenných prostorů, z nichž každý obsahuje třídy a další typy, jež sdílejí společný účel. Kupříkladu většina ze správy oken API Windows je zapouzdřena ve jmenném prostoru *System.Windows.Forms*, kde nalezneme třídy představující okna, dialogová okna, nabídky a další elementy obecně používané v aplikacích mající grafické uživatelské rozhraní. Jiný samostatný jmenný soubor, označený jako *System.Collections*, obsahuje třídy, které představují hashovací (asociativní) tabulky, pole

s proměnnou velikostí a jiné datové kontejnery. Další jmenný prostor, *System.IO*, zahrnuje třídy pro operace vstupu a výstupu se soubory. Kompletní seznam všech těchto jmenných prostorů nacházejících se ve FCL lze nalézt jako součást online dokumentace sady SDK rámce .NET Framework. Úkolem každého začínajícího programátora tak je se s těmito jmennými prostory seznámit. Naštěstí je však FCL tak rozsáhlá a všezahrnující knihovna, že drtivá většina programátorů ji nikdy nebude muset umět zvládat celou. [14]

5 Analýza informačního systému pro restaurační a barová zařízení

5.1 Požadavky

Požadavky budou rozděleny do několika částí, jelikož výsledný informační systém bude mít také více zaměření. První část by se měla zaměřovat na kontrolu, zpracování a uchování denních inventur (rozpisů). Další část by měla zajišťovat tzv. knihu příjmů a výdajů. Jednou ze zamýšlených součástí informačního systému by měla být kniha norem, ve které se budou uchovávat jednak recepty jednotlivých pokrmů a jednak množství jednotlivých použitých surovin. Rozpis směn zajišťuje zadávání a úpravu nasazování jednotlivých pracovníků do jednotlivých směn v měsíci.

Jednotlivé požadavky byly získány především studiem dosavadních systémů doplněné o interview s vedoucím pracovníkem. Dotazy pak byly kladeny i zaměstnancům, kteří budou také systém využívat.

Studium dosavadního systému spočívalo v práci se skutečnými daty a zadáváním do systému. To umožnilo hlouběji proniknout do dané problematiky a navrhnout případné optimalizace a vylepšení v novém informačním systému.

V další fázi proběhlo sledování vedoucího pracovníka při práci v systému, aby se zjistily případné další možné optimalizace. Pozorováním pracovníků při inventarizaci zboží, které probíhá obvykle v brzkých ranních hodinách, bylo zjištěno, že pracovníci dělají poměrně hodně početních chyb, které vedou k nepřesnostem.

5.1.1 Evidence zboží

Evidencezboží se využívá v barových a restauračních zařízeních ke kontrole a evidenci prodaného, dodaného, odepsaného zboží a jiných položek.

Evidence zboží může probíhat několika různými způsoby. Klasickou papírovou formou, kdy na konci předem daného období (v našem případně dny, případně při předání směny) se sepíše seznam zboží a doplní se konečný stav zboží na skladu včetně dodaného a odepsaného zboží. Z tohoto lze spočíst tržbu od data, kdy byl naposledy zapsán poslední stav. Nevýhoda tohoto systému je, že u zařízení s větším množstvím sortimentu je procedura poměrně zdlouhavá. Kladem však je poměrně přesné zjištění stavu zboží na skladě.

Další možností je evidovat zboží pomocí pokladních systémů. Pokladních systému je v dnešní době velké množství. Od jednoduchých, až po pokročilé systémy se skladovým hospodářstvím. Problém však nastává v tom, že se do pokladního systému nezadají všechny prodané položky, nebo se

špatně zadají. Pak může nastat situace, že pokladní systém eviduje jiné množství zboží, než je skutečně na skladě. Nevýhodou může být také cena, která u kvalitnějších zařízení je vysoká, což u malých provozoven nepřipadá v úvahu. Pokladní systémy se skladovým hospodářstvím, pak zajišťují jednak stav prodaného zboží na pokladně a jednak fyzický stav zboží na skladě, který je možno zadávat po inventuře, a tak porovnávat pohyb zboží mezi pokladnou a skladem.

Kombinací předchozím dvou možností eliminujeme většinu špatných vlastností. Pomocí pokladního systému lze zjistit tržbu za dané období. Pro kontrolu stavu skladu se pak provede „klasický“ rozpis, který je pak možné dělat v delších časových intervalech.

Rozpis v klasické tištěné formě obsahuje několik sloupců, které osoba provádějící inventuru vyplní. Jedná se především o název zboží, jednotku, cenu za jednotku, poslední stav, dodáno, režie, celkem (suma předchozích 3), zůstatek, prodej a částka. Některé sloupce mohou být vyplněny před začátkem inventury. Tím se usnadní práce osobě provádějící inventuru. Někdy toto může být nevýhodné z hlediska zaměstnavatele/provozního, protože zaměstnanci se mohou snažit úmyslně zaměnit zůstatky.

5.1.2 Peněžní deník

Jedná se o evidenci příjmů a výdajů. Nejedná se o účetní evidenci, kde se evidují transakce hlavně z hlediska daní. Eviduje se tok financí na jednotlivých provozovnách. Z peněžního deníku pak lze vyčíst kdy a na co byla daná částka poskytnuta, nebo naopak odkud byla přijatá. Souhrnem těchto transakcí je pak možno zjistit aktuální, nebo předchozí finanční stavy. Zadávány jsou tedy například údaje tržba, nákup zboží, záloha, výplata atp.

5.1.3 Kniha norem

Obecně lze knihu norem zakoupit v knihkupectví. Jedná se o knihu, ve které je sepsáno mnoho běžně užívaných receptů. Recepty jsou určeny především pro restaurace, jelikož jednotlivá jídla jsou zde obvykle pro 10 osob. U surovin se evidují dvě váhy. Jednak hrubá váha a jednak čistá váha. Hrubá váha je vždy vyšší, nežli čistá. Je to způsobeno tím, že hrubá váha udává hmotnost suroviny před zpracováním (při nákupu) a čistá váha pak udává hmotnost po zpracování (např. brambory po oškrábání).

Avšak ne všechny recepty jsou zde zobrazeny a mohou se také lišit od připravovaných kuchařem. Proto je tato kniha brána spíše jako referenční. Evidovat množství jednotlivých surovin receptů vyžadují české zákony. Každý podnik si vytváří vlastní evidenci norem, podle kterých se pak musí jednotlivé pokrmy připravovat.

5.1.4 Rozpis směn

K zajištění správné organizace provozu je rozpis směn nezbytným nástrojem každého podniku. Rozpis směn určuje kdo, kdy a kde bude pracovat. Mimo to může také evidovat dovolené a nemocenské. Tímto nástrojem zajistíme jednak předání informace o pracovním místě a době zaměstnanci, ale také snadněji zjistíme, zda na každou směnu přijde požadovaný počet zaměstnanců.

5.2 Stávající systém

Provozovny, pro které bude implementován novy informační systém, doposud využívají systém založený na produktu Microsoft Excel. To je nevýhodné z několika hledisek. Největší nevýhoda spočívá především ve značné redundanci dat při uchovávání předchozích stavů - vytvoří se kopie celé tabulky, nikoli jenom část, která se změnila. Problém je pak také v procházení a hledání dané tabulky, nebo řádku v historii.

5.2.1 Evidence zboží

Evidence zboží je vytvořena jako tabulka v programu Microsoft Excel. Tabulka je obdobná tištěné formě, je přidán jen sloupec EAN (identifikační číslo zboží v pokladním systému). Uživateli jsou přístupny všechny buňky neobsahující vzorce. Vzorce obsahují sloupce celkem (suma poslední stav, dodáno a režie), prodej (celkem mínus zůstatek) a částka (prodej krát částka za jednotku). U vzorce pro prodej však může nastat inverzní případ, tedy zůstatek mínus celkem, to nastane v případě, že odečítáme stav zboží z počítadla. Počítadlo se při prodeji zboží inkrementuje, kdyžto při prodání zboží bez počítadla se stav sníží. Na konci každého listu je celková částka za prodané zboží na dané stránce. Na poslední stránce je pak navíc celková utržená částka.

Převod zůstatku na poslední stav je pak zajištěn pomocí makra. Toto makro přepíše buňky posledního stavu buňkami zůstatků. Následně se také vytváří dvě kopie listu. Oba obsahují všechny sloupce, ale v prvním nejsou vyplněny od sloupce dodáno až po celkovou částku. Ve druhém pak chybí ještě hodnoty sloupce poslední stav. Oba tyto listy slouží k vytištění a k následující inventuře.

5.2.2 Peněžní deník

Tabulka listu je rozdělena na levou a pravou stranu. Na levou stranu se zadávají výdaje a na pravou stranu příjmy. Jednotlivé položky jsou seskupovány vzestupně podle data. Na konci každé stránky je zobrazen aktuální stav financí.

5.2.3 Kniha norem

Doposud se využívají dvě varianty. Tabulky v Excelu, které zobrazují jednotlivé recepty. A druhá varianta, která využívá pokladního systému Vectron se skladovým hospodářstvím Comarr, který spravuje jednak jednotlivé kalkulace, ale obsahuje také bourací listy a propojení se skladem a pokladním systémem Vectron.

5.2.4 Rozpis směn

Rozpis směn je prvotně zadáván vedoucím/provozním do šablony v Excelu s přihlédnutím na požadavky zaměstnanců (dovolená, nemocenská, atp.) a poté vytištěn a předán zaměstnancům. V případě nepředvídatelných situací se pak změny provádí pouze na vytištěném rozpisu.

5.3 Neformální specifikace

5.3.1 Rozpis

Přístup k informačnímu systému rozpisu budou mít tři druhy uživatelů. Jedná se o uživatele vedoucí, provozní a zaměstnanec. Provozní a zaměstnanec mají v této části stejná práva, avšak vedoucí může tato práva zaměstnanci omezit (viz. Uživatelská práva). Vedoucí má možnost přidat nový druh zboží, následně jeho editaci a odstranění. Každé zboží by mělo jít snadno vyhledat a následně s ním pracovat. U všech druhů zboží by měla být možnost zadat minimální a maximální stav skladu. Při překročení této hranice by měl být varován vedoucí nebo provozní. Po uzávěrce by měla být možnost exportovat výslednou částku a informaci o ní do peněžního deníku, kde se vloží do sloupce příjmů. Zaměstnanci mají možnost vkládat pouze záznamy o novém stavu skladu, tedy dodáno, režie a zůstatek (případně prodej). Tuto možnost mají také vedoucí a provozní.

5.3.2 Peněžní deník

Do peněžního deníku by měl mít přístup jen vedoucí a provozní. Zaměstnanci nemají do této části systému vůbec přístup. Peněžní deník obsahuje dva druhy záznamů: příjem a výdaj. Tyto údaje lze přidávat, editovat a odstraňovat a také zpětně prohlížet a zobrazovat za dané období, případně podle názvu záznamů.

5.3.3 Kniha norem

Aby bylo možné zadat kalkulaci, je potřeba mít vyplněný číselník surovin. Suroviny je možné přidávat, editovat a odstraňovat. U surovin je také možnost zadat cenu za jednotku. Nová kalkulace, neboli norma, se vytváří seskupením používaných surovin a přiřazení množství. Je zde také možnost

zadat počet porcí, případně cenu, za kterou se bude prodávat. U každé kalkulace je možné zjistit hmotnost porce, případně jeho cenu, pokud jsou zadány ceny surovin. Jsou-li zadány ceny suroviny a ceny kalkulací, pak bude systém varovat při změně ceny suroviny, že cena kalkulace překročila minimální mez. Zaměstnanec, obvykle kuchař, má možnost zobrazovat jednotlivé kalkulace.

5.3.4 Rozpis směn

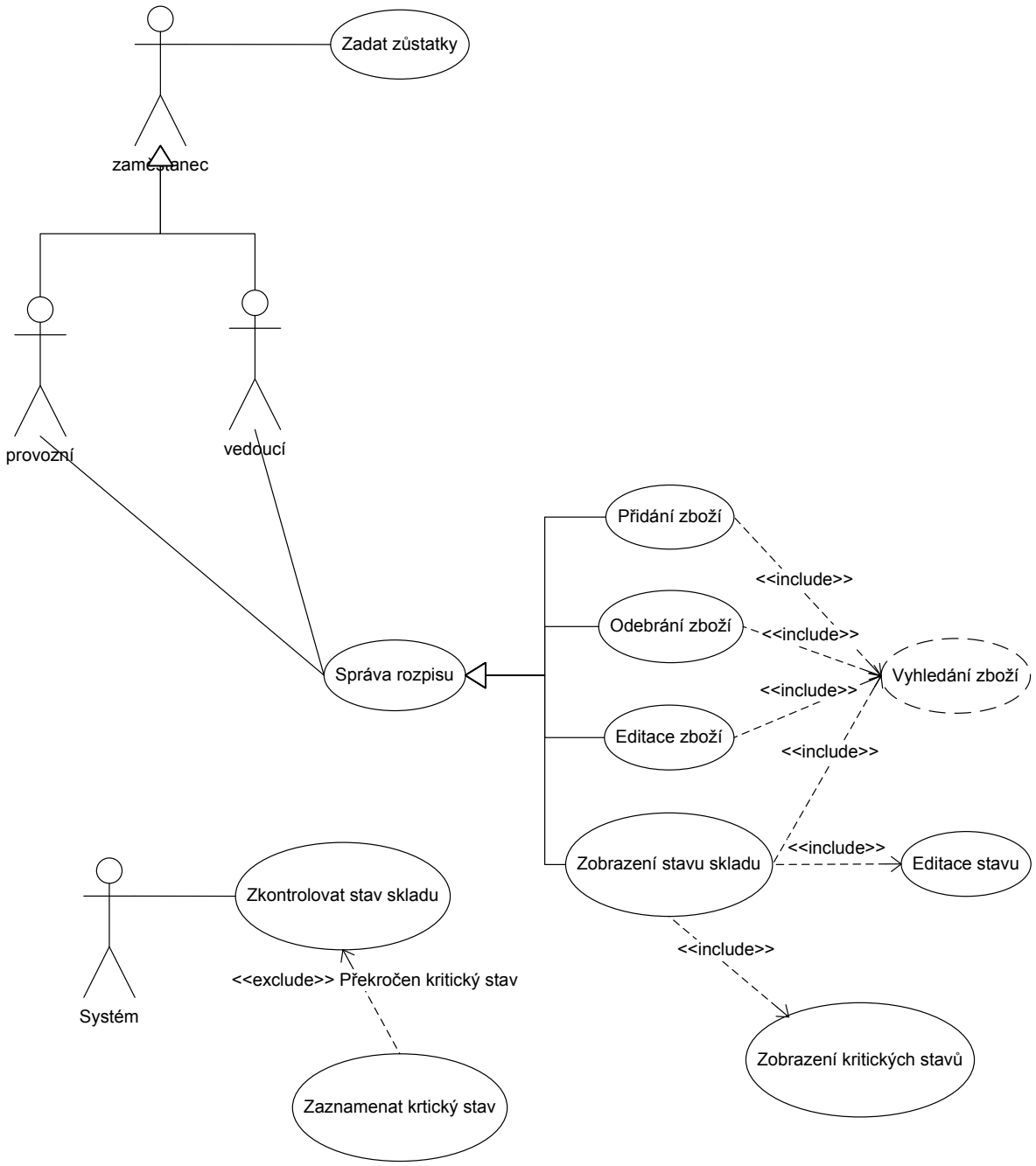
Provozní a vedoucí mají možnost přiřazovat zaměstnance na jednotlivé směny. Měly by mít také možnost zadávat omezení, která zamezují přiřazení zaměstnance na některé směny. Jednotlivé směny jsou také vytvářeny těmito uživateli (ranní, odpolední, noční, atp.). Zaměstnanci mají možnost nahlížet na rozpisy směn a zadávat požadavky na směnu. To znamená, že má možnost zadat do systému na aktuální den, kdy by chtěl volno, případně jakou směnu. Vedoucí a provozní má pak možnost nahlédnout na tyto požadavky a buď akceptovat tyto požadavky, nebo k nim nepřihlížet. Systém by měl zamezovat nasazení zaměstnance na více směn, které by se překrývaly a nebyly v souladu se Zákoníkem práce.

5.3.5 Uživatelské účty

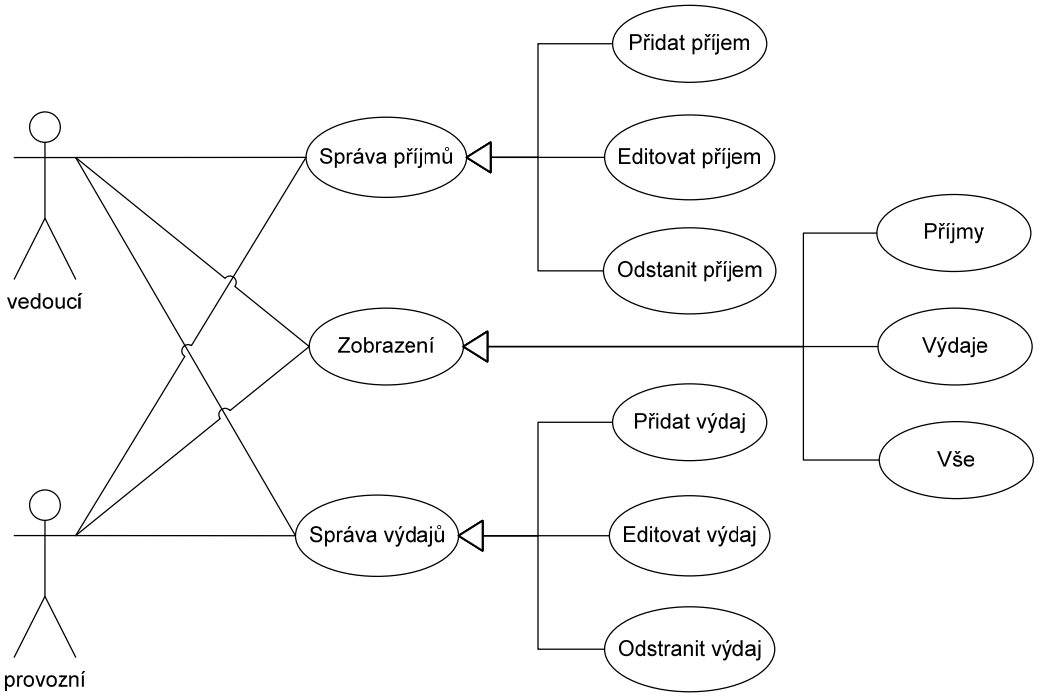
Každý uživatel má možnost změnit si osobní údaje svého účtu. Možnost přidání nového uživatele, editace a odstranění má vedoucí a provozní. Vedoucí má oproti provoznímu možnost spravovat účty provozních.

5.4 Diagramy případu užití

Jednotlivé diagramy byly pro přehlednost rozděleny na menší části podle oblasti, kterou se zaobírají. Tedy na digramy rozpis, peněžní deník, kniha norem a rozpis směn a uživatelská práva.

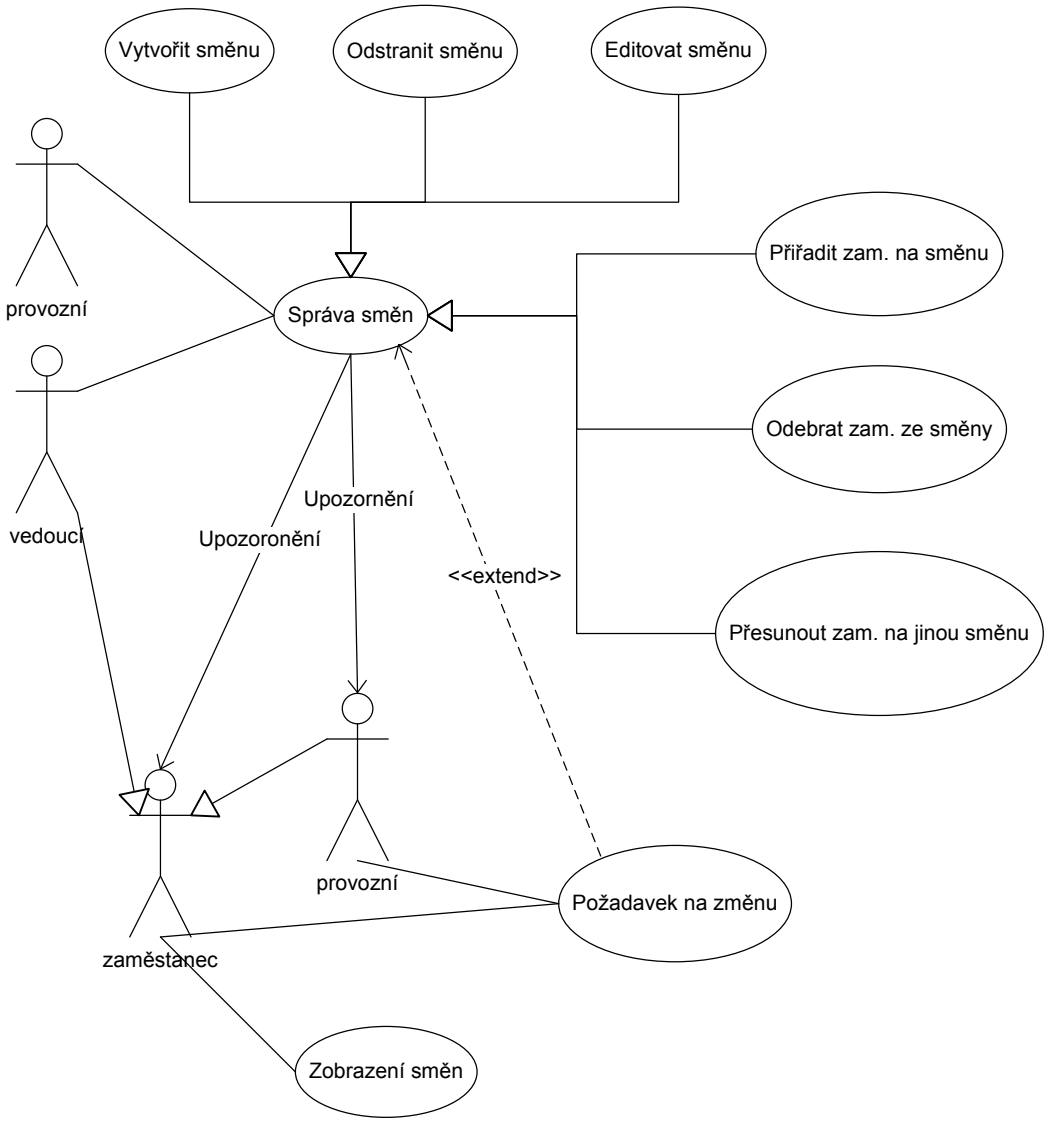


Obrázek 5.1 rozpis

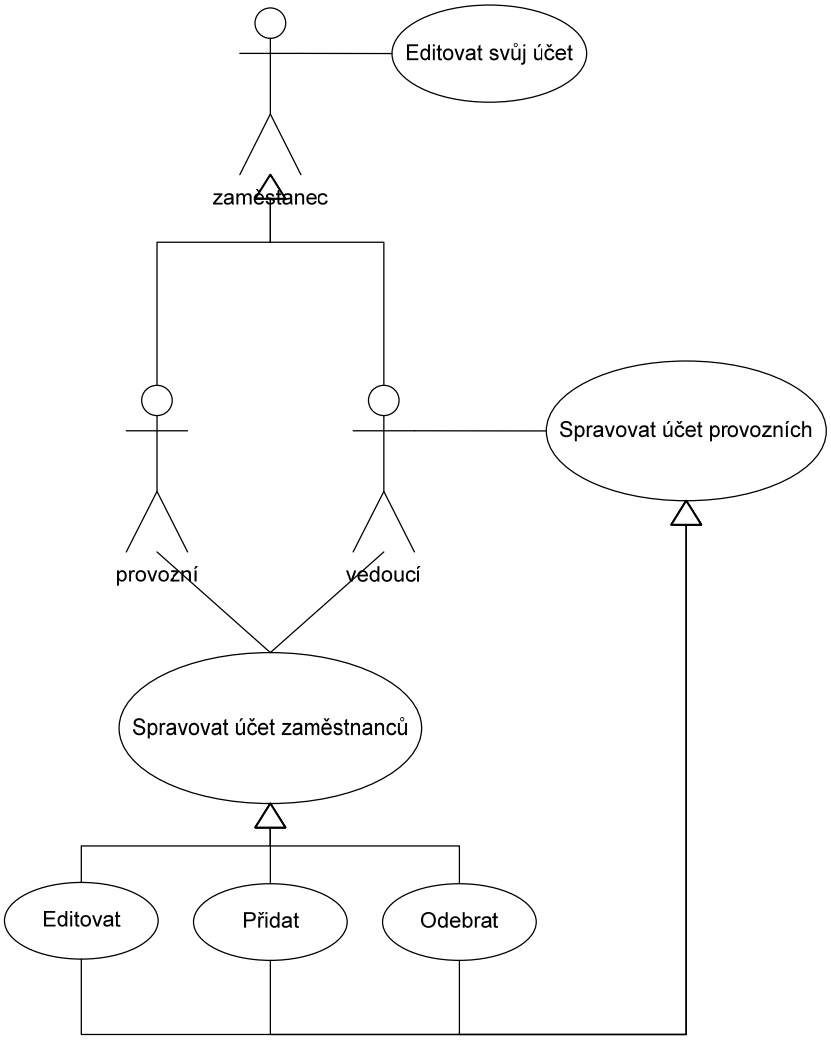


Obrázek 5.2 peněžní deník

Obrázek 5.3 kniha norem



Obrázek 5.4 rozpis směn



Obrázek 5.5 uživatelská práva

5.5 ER diagramy

Jednotlivé diagramy jsou rozděleny, obdobně jako předcházející diagramy případu užití, na jednotlivé části. Tímto se zvýší přehlednost a čitelnost.

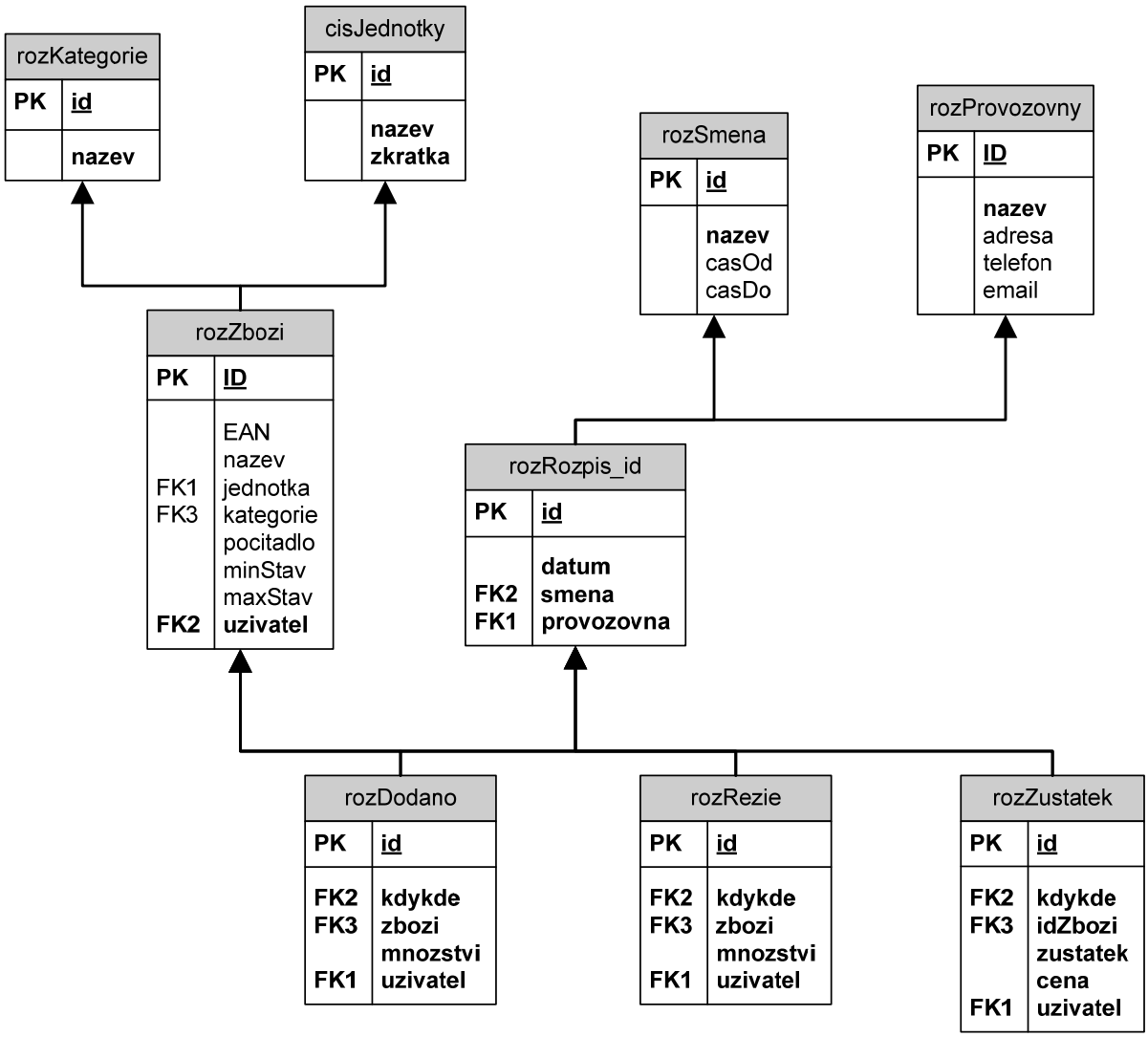
Datový model pomocí ER diagramů byl navržen tak, aby splňoval minimálně třetí normální formu.

5.5.1 Rozpis

Jednotlivé druhy zboží lze přiřazovat k různým provozovnám. Dále je nezbytné uchovávat pro každý druh zboží jeho stav na skladu po uzávěrce. Mimo stavu skladu je také potřeba uchovávat informace o dodaném zboží, ale také o zboží, které bylo odepsáno (např. u zboží, které prošlo minimální dobou spotřeby). Aby bylo dosaženo oddělení skladů jednotlivých provozoven a stavu zboží pro jednotlivé

směny při zachování minimálně třetí normální formy, byla vytvořena tabulka rozRozpis_id, která odděluje zmiňované části.

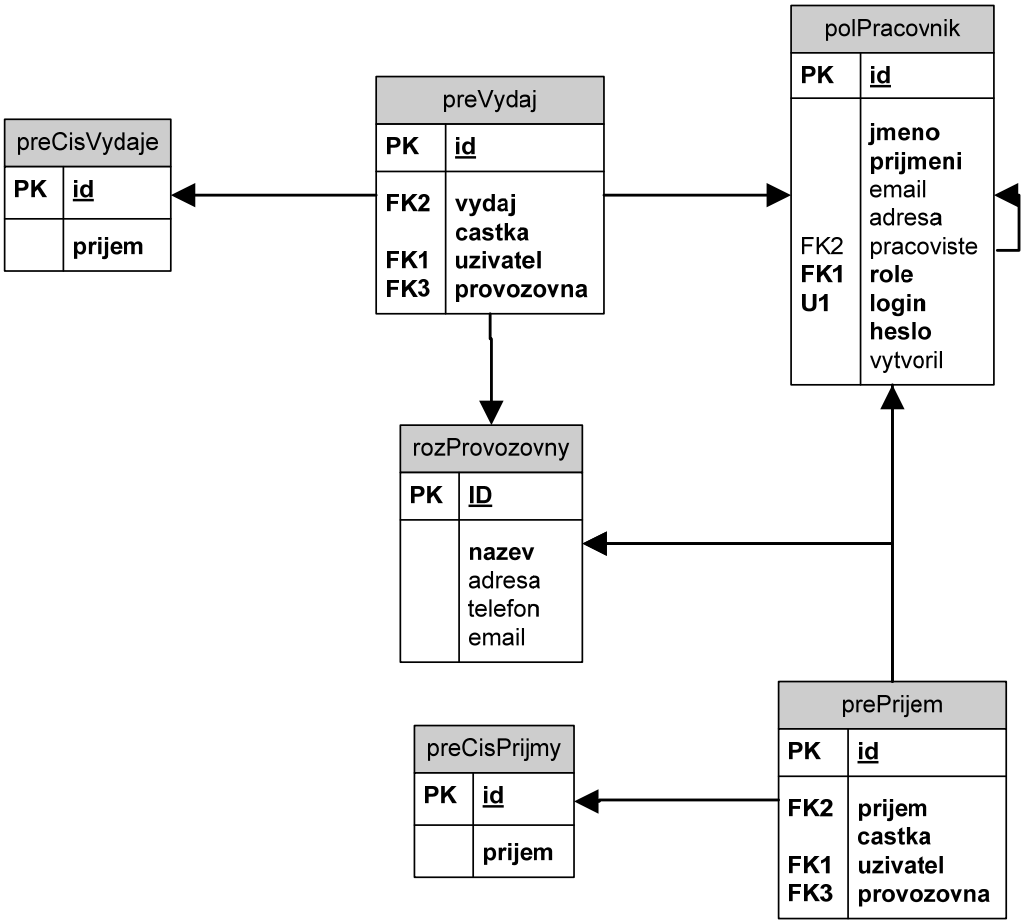
V diagramu chybí propojení s diagramem uživatelů. Tabulka polPracovník, která je propojena relací s tabulkami *rozRezie*, *rozDodano*, *rozZustatek* a tabulkou *rozZbozi*. Tyto tabulky obsahují atribut *uzivatel*, udávají informace o uživateli, který vložil daný záznam, případně záznam naposledy editoval.



Obrázek 5.6 rozpis

5.5.2 Peněžní deník

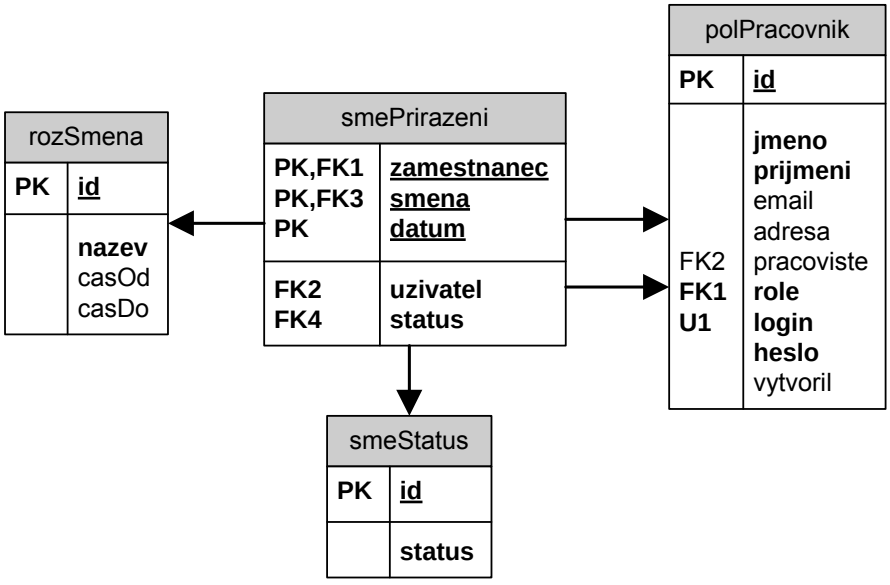
Peněžní deník se skládá ze dvou základních tabulek – příjmů a výdajů. Pro odstranění redundance při zadávání stejných popisů příjmů a výdajů byly vytvořeny dvě pomocné tabulky, které budou sloužit jako slovníky.



ER diagram - peněžní deník

5.5.3 Rozpis směn

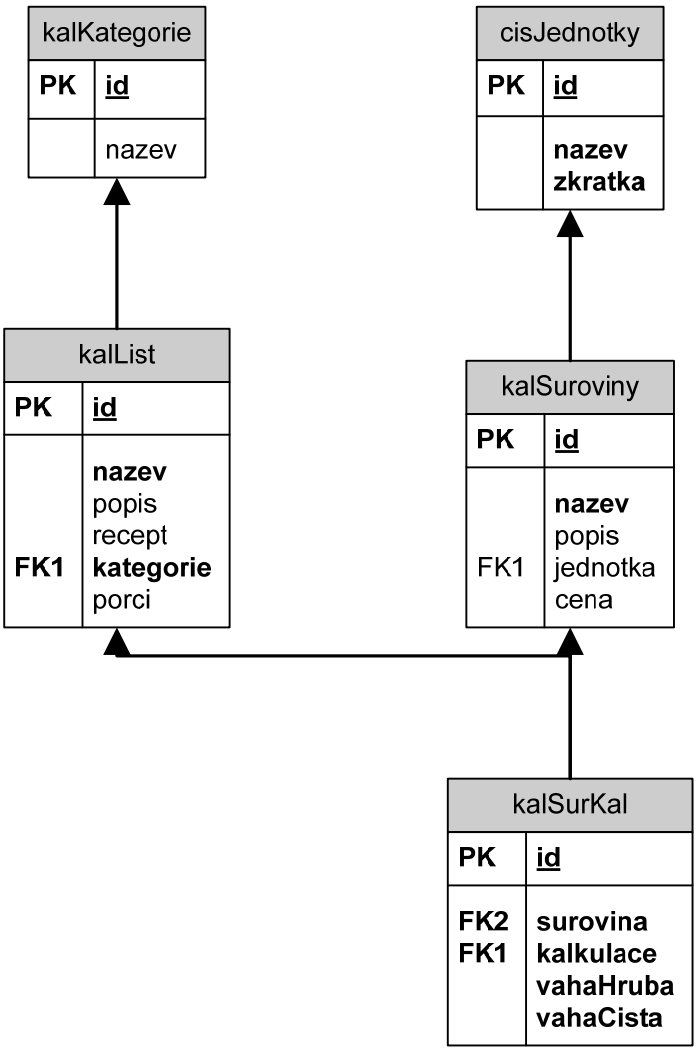
Základ tvoří tabulka *smePrirazeni*. V této tabulce se přiřazuje zaměstnanec na směnu. Atribut *status* zde bude udávat, jestli se jedná o požadavek zaměstnance o danou směnu, nebo se jedná přiřazení zaměstnance na směnu.



Obrázek 5.7 rozpis směn

5.5.4 Kniha norem

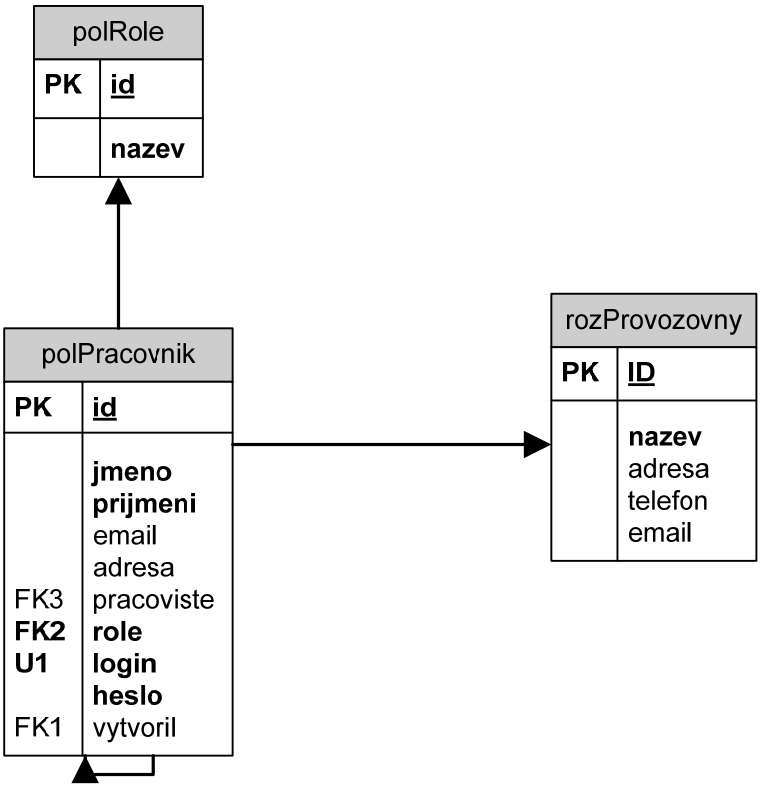
Kniha norem obsahuje dva základní číselníky. Jedná se o měrné jednotky a seznam surovin. Suroviny jsou pak přiřazovány ke kalkulačním listům (vytvoření kalkulace).



ER diagram - kniha norem

5.5.5 Uživatelská práva

Všichni uživatelé jsou uloženi v tabulce *polPracovnik*. Atribut *role* této tabulky pak určuje, jakou roli bude mít daný uživatel. Zaměstnance a provozní je pak možno přidělit k některé z provozoven.



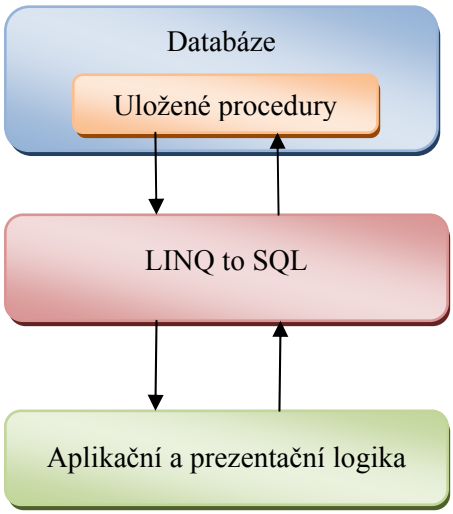
Obrázek 5.8 uživatelé

6 Návrh řešení

6.1 Návrh architektury

Pro správný návrh systému je nejprve nutné rozdělit jednotlivé funkcionality systému do vrstev, které tvoří jako celek výsledný systém. Každá vrstva zastává ve vícevrstvé architektuře pevně danou funkcionalitu. Tím, že se rozdělí funkcionalita do vrstev, se stává systém přehlednější, srozumitelnější, znovupoužitelný a rozšiřitelný. Předchozí je problém zajistit u monolitických aplikací.

Navrhovaný systém je rozdělen do tří vrstev: vrstva datová, vrstva .LINQ a nakonec vrstva aplikační a prezenční logiky. Vrstva LINQ to SQL zajišťuje provedení požadovaných dotazů v databázi a do jisté míry tak odděluje databázovou a aplikační vrstvu. Prostřední vrstva tak zajistí nezávislost aplikace na zvolené databázi (doposud pracuje LINQ to SQL s databází SQL Server, do budoucna se počítá také s Oracle a MySQL). To znamená, že při změně databázového systému není nutné zasahovat do aplikačního kódu.



Obrázek 6.1: Schéma systému

LINQ to SQL zajišťuje mimo jiné také mapování relačních dat na objekty. Každý objekt v aplikaci má své vlastnosti a metody, které umožní jeho správu. V našem případě se jedná o mapování uložených procedur do vytvořených entit. Tímto je zajištěno oddělení nejen od typu databáze, ale i její struktury.

V databázové vrstvě je podvrstva uložené procedury, která zajišťuje další oddělení databáze od vrstvy LINQ to SQL. Při změně struktury databáze, při zachování vstupních parametrů a výstupní struktury, lze zajistit změnou uložené procedury stavu, kdy se navenek bude chovat databáze pořád stejně a není potřeba měnit kód na ostatních vrstvách.

6.2 Návrh databáze

Pro správnou funkci je potřeba navrhnout databázi tak, aby obsahovala co nejméně redundantních dat, ale zároveň aby dotazy nad touto databází netrvaly neúměrně dlouho. Při návrhu je také potřeba dbát na datové typy, aby nebyly zbytečně předimenzované, ale také aby mohli pojmout data, která budou uchovávat.

Po vytvoření struktury databáze následuje fáze vytváření uložených procedur. Je potřeba vytvořit takové uložené procedury, které zajistí zpřístupnění potřebných záznamů na základě vstupních parametrů, úpravu záznamů a také odstranění záznamů. Mimo uložené procedury je možné také vytvořit databázovou funkci, které má jednu návratovou hodnotu. Tato možnost je výhodnější, pokud výsledkem dotazu bude právě jedna hodnota daného datového typu.

Databáze byla navržena v 3. normální formě, což zajistí značnou eliminaci redundantních dat a zamezí nekonzistentním závislostem.

Struktura databáze viz příloha.

7 Implementace

7.1 Výběr databázového systému

Existuje několik vhodných databázových systémů pro vývoj, údržbu a správu databáze. Zvolil jsem databázový systém Microsoft SQL Server. Tento systém jsem zvolil z několika důvodů. Jedním z nich je podpora ze strany .NET Frameworku a Visual Studia, tak že dokáže relativně snadno zjistit databázovou strukturu a použít ji. Velkým plusem je také možnost použití uložených procedur pro dotazování. Dalším důvodem je, že lze použít SQL Server Express, který je zdarma. Tato edice je pro daný systém zcela dostačující.

Při vývoji jsem použil tři různé verze SQL Serveru. Jedná se o tyto:

- SQL Server 2005
- SQL Server Express
- SQL Server Code Name „Katmai“

Při používání žádného z uvedených serverů jsem nezjistil žádné potíže při vývoji, ani při samotném používání aplikace. Poslední jmenovaný, SQL Server Code Name „Katmai“, je poslední veřejnou testovací verzí SQL Serveru 2008.

Jednou z velkých výhod tohoto databázového systému je možnost použít uložené procedury. Při výběru byla váha přikládána i k tomuto, protože uložené procedury snižují přenos komunikačního kanálu u rozsáhlejších dotazů a také jsou optimalizované na danou databázi podle prováděcího plánu.

SQL Server nabízí několik nástrojů pro vývoj, optimalizace a správu serveru. Tyto nástroje umožní včas zjistit chyby na různých úrovních životního cyklu databáze.

7.2 Výběr operačního systému

Výběr operačního systému úzce souvisí s výběrem databázového systému. Jelikož všechny klientské počítače a případné servery, na nichž se bude systém používat, mají nainstalován operační systém Windows XP, případně Windows Vista, byla volba daná. Aplikace je stavěna na platformě .NET Framework 3.5, což může být jediný problém k úspěšnému zprovoznění. Ke spuštění aplikace je nutné mít nainstalován tento framework. Z předchozího plyne, že nelze brát v úvahu alternativní operační systémy.

Spustitelný soubor .NET Frameworku je, obdobně jako programy napsány v Javě, překládán do mezikódu. Tento mezikód se překládá do finální podoby na klientské stanici před spuštěním nebo při instalaci. Funkce mezikódu je výhodná v tom, že kód je optimalizován pro daný stroj, na kterém

běží. To může mít za následek, že cílová aplikace může být rychlejší kompilovaná do mezikódu, než do binární spustitelné podoby.

7.3 Výběr vývojového prostředí

Pro vývoj aplikací a služeb v .NET Frameworku existuje několik vývojových prostředí. Vybral jsem Visual Studio, které je asi nejlepší po stránce psaní kódu, ladění, práci s datovými zdroji, aj. Využívá také poměrně kvalitní funkci *IntelliSense*, která nabízí programátorovi možné dokončení klíčových slov, metod apod.

Zpočátku implementace bylo použito Visual Studio 2005 s nadstavbou „*Orcas*“, což je rozšíření o nové assembly .NET Frameworku 3.5. Po uvedení Visual Studia ve verzi 2008 jsem přešel na tuto verzi. Obrovské pozitivum při vývoji aplikací ve Visual Studiu je jeho svižnost a relativně malé nároky na paměť.

Komponenty lze vytvářet v .NET Frameworku v několika jazycích. Základními jazyky .NET Frameworku jsou C#, Visual Basic, C++ a J#. Výběr programovacího jazyka nemá výraznější vliv na chování a vizuální vzhled aplikace, proto jsem vybral jazyk, který je mi nejbližší, a to C#. Ve verzi .NET Framework 3.5 je implementována podpora pro C# ve verzi 3.0 a nižší.

Ve verzi jazyka C# verze 3.0 jsou dvě základní možnosti, jak přistupovat k databázovému systému. Jedná se o ADO.NET a LINQ to SQL. Zvolil jsem poměrně novou moderní technologii LINQ to SQL pro její přehlednější a mnohem intuitivnější správu, zpracování a zobrazení dotazovaných dat.

7.4 Návrh uživatelského rozhraní

Návrh uživatelského rozhraní je nesporně jednou z nejdůležitějších částí návrhu informačních systémů. Informační systém, který je dobře implementován a má požadovanou funkčnost, ale má nevhodně navržené uživatelské rozhraní, může vést až k nepoužitelnosti celého systému. Proto je potřeba vytvořit takové rozhraní, které bude možné ovládat co možná nejsnáze. V tomto hraje velkou roli také rozložení prvků na formulářích a případně návaznost mezi jednotlivými formuláři.

.NET Framework od verze 3.0 má dvě možnosti, jak se bude zobrazovat formulář. Jedná se o starší Windows Forms, které využívají k vykreslování oken GDI+ a o novou technologii WPF (Windows Presentation Fundation), která je zcela vektorová a dovoluje vytvářet design formulářů, který byl ve Windows Forms jen obtížně dosažitelný.

Pro cílovou aplikaci jsem použil starší Windows Forms, jelikož novinky, které přináší WPF by nebyly využity. WinForms je pro danou aplikaci dostačující.

Aplikace se skládá ze čtyř na sobě nezávislých částí, které by mohly být implementovány jako čtyři oddělené aplikace, nebo jako jediná, která by plnila funkci rodičovského formuláře. Přímo

se vybízí možnost využiti MDI (Multi Document Interface) formuláře. MDI umožňuje v rámci jednoho formuláře otevřít několik dokumentů zároveň a plynule mezi nimi přecházet. Tyto dokumenty (formuláře) pak zastávají funkci potomků.

7.5 Struktura aplikace

Samotná aplikace je strukturována na čtyři části, které jsou dosažitelné v hlavním okně informačního systému. Pro spuštění potřebného modulu se může uživatel rozhodnout, zda otevře aplikaci pomocí tlačítka v nástrojové liště, nebo přes menu, kde najdeme také podrobnější možností spuštění daného modulu.

Po spuštění aplikace je vyžadováno vyplnění uživatelského jména a hesla. Při zadání neplatné kombinace je uživatel požádán o opětovné zadání, jinak je aplikace uzavřena. Při správném přihlášení je vyhodnocen přihlášený uživatel a zjištěna role, kterou má v systému. Na základě této role je mu umožněn přístup do sekcí, na které má právo, případně zápis nebo editace údajů. V systému jsou vytvořeny tři role:

- Vedoucí (administrátor): má práva vstupu do všech sekcí a provozoven
- Provozní: má právo vstupu do všech sekcí, ale pouze v rámci provozovny, ve které je zaměstnán; má možnost vytvářet zaměstnance v rámci své provozovny
- Zaměstnanec: Má povolen vstup pouze do sekce Rozpis (pouze zadávání dat inventury), Knihy norem a rozpisu směn pro vlastní osobu

Vedoucí a provozní musí po přihlášení zvolit provozovnu, se kterou bude pracovat. Vždy může být aktivní pouze jedna provozovna. Uživatel ji má možnost změnit; při změně jsou zavřeny všechny podokna aplikace, aby nedocházelo k chybnému zadávání údajů.

7.5.1 Evidence zboží

Modul rozpis byl koncipován tak, aby byl co možná nejpodobnější a papírové formě a současné podobě – tabulky v MS Excel. V tabulce se zobrazuje všechno zboží, které bylo zavedeno na dané provozovně. U každého zboží je zobrazen v řádku jeho detail, včetně posledního stavu, ceny a kategorie. Detail zboží lze zobrazit kliknutím na název, nebo EAN a případně jej změnit. Při dodání, nebo režii zboží je postup obdobný.

Zadávání zůstatku lze provést přímo v tabulce změnou hodnoty. Nový stav lze zadat buď zadáním množství prodaného zboží, nebo jeho zůstatek. U některých druhů zboží, zejména u těch, která jsou evidována počítadlem, je počítání opačným směrem (při prodeji zboží je zůstatek vyšší, nežli původní). Tato možnost lze zvolit v detailu zboží. U každé inventury lze také zadávat částku, za kterou se účtuje dané zboží.

Při jakékoli změně v tabulce jsou data daného záznamu přepočítána a také aktualizován záznam o celkové částce. U každého zboží je možno zadat minimální a maximální množství, které je na skladu. Při překročení hranice je záznam obarven na příslušnou barvu, jako upozornění uživateli. Záznam je také obarven v případě, že zůstatek se dostane do záporných hodnot.

Jednotlivé záznamy je možné filtrovat, díky zabudovanému filtru. Lze vyhledávat podřetězce názvu zboží, číslo EAN. Mimo předchozí lze také vybrat pouze kategorii zboží, kterou chceme zobrazit. Záznamy, které překračují hranici minimálního, případně maximálního stavu lze také filtrovat zaškrtnutím příslušných zaškrtávacích polí.

7.5.2 Peněžní deník

Formulář příjmů a výdajů je rozdělen na dvě části – levou a pravou. V levé části se zadávají do tabulky výdaje a do pravé příjmy. Pro názvy příjmů a výdajů byly vytvořeny číselníky, jelikož názvy se poměrně často opakují. Tímto se zamezí do jisté míry redundanci dat.

Po otevření formuláře se načte z databáze aktuální měsíc. Ve formuláři lze změnit rozsah vstupních dat. Při této akci se načtou z databáze potřebné záznamy. Při jakékoli změně, či přidání nových záznamů se aktualizuje celková částka a částka za zobrazené období.

7.5.3 Rozpis směn

Rozpis směn umožňuje vedoucímu nebo provoznímu přidat zaměstnance na směnu. To znamená, že je vybrán zaměstnanec, datum a směnu, na kterou má nastoupit. Tito uživatelé mají možnost si také prohlížet žádosti zaměstnanců na danou směnu, nebo volno. V uživatelském rozhraní je aplikován filtr, který umožňuje zobrazovat pouze potřebné záznamy.

Zaměstnanci mají možnost pouze prohlížet směny a zadávat požadavky na změnu, či přidání směny.

7.5.4 Kniha norem

Formulář je rozdělen do vodorovně do tří základních částí. Vrchní zobrazuje seznam vytvořených kalkulací, prostřední, která zachycuje detail kalkulace, jako je popis kalkulace a recept. Ve spodní části nalezneme tabulku se seznamem a množstvím použitých surovin.

V první zmíněné tabulce se vytváří hlavička receptury, případně její editace. Při výběru řádků (receptury) se zobrazí ve spodní části detail. To znamená, že ve spodní tabulce budou zobrazeny suroviny, které daná kalkulace obsahuje. Suroviny je možné přidávat, či odstraňovat z tohoto seznamu.

Aby bylo možné zařadit surovinu do kalkulace, je nutné ji nejprve vytvořit. Suroviny se ukládají do číselníku, takže je zamezeno redundanci.

7.5.5 Tiskový modul

Vybrané vývojové prostředí nabízí tisk pouze pomocí GDI+, to znamená, že pokud chceme tisknout data ve strukturované podobě, je vhodné vytvořit modul, který toto zajistí. Nejvhodnějším způsobem, jak tisknout data z předchozích modulů je v tabulkové formě.

Jednou z možností, jak vytisknout tabulku je vytisknout obrázek komponenty DataGridView, ve které jsou zobrazeny data, ale tato možnost byla zavrhnuta díky slabému vizuálnímu výstupu a minimální možnosti ovlivnění výsledného zobrazení.

Druhou a použitou metodou je vytvoření modulu, který vytvoří tabulku z grafických primitiv na základě vstupních dat. Tisk se tak sestává z hlavičky, patičky a těla formuláře. Tisknout data lze jak v přehledné tabulkové formě, tak ve formě řádkové, kdy je detail záznamu vždy na samostatném řádku.

Tiskový modul využívá předdefinovaných dialogů .NET Frameworku pro nastavení stránky, náhled tisku (viz příloha) a pro tisk dokumentu.

8 Závěr

Cílem této diplomové práce bylo seznámit se s metodami určování požadavků na software a jejich specifikace. Z těchto metod vybrat vhodné a aplikovat je na reálný informační systém pro restaurační a barová zařízení. Z navrhnuté architektury vytvořit informační systém, který bude plně funkční a připraven k nasazení do provozu.

V první části jsem se zabýval softwarovým procesem a jeho typy. V následující části jsem se věnoval jednotlivým etapám životního cyklu a také modelům životního cyklu, jako jsou např. vodopádový model nebo iterativní model. Důležitou části pak je kapitola analýza a specifikace požadavků, která se zabývá především určením, získáním, uspořádáním a validací požadavků. Tato část může velmi usnadnit a zrychlit práci v dalších fázích při správném určení všech požadavků na daný systém. Při nesprávných nebo neúplných požadavcích nastávají chyby v následujících fázích, pak je potřeba se vracet do předchozích fází a tyto chyby opravit.

V další fázi jsem se zaměřil na sběr požadavků ve firmě. Požadavky jsem získal jednak z nastudování dosavadního systému, ale také aktivním pozorováváním budoucích uživatelů systému při jejich práci. Ze získaných požadavků jsem pak vytvořil diagramy případů užití pro všechny části systému. Pro vytvoření datového modelu jsem pak vytvořil ER diagramy, které jsou pro relační databáze vhodnější (lze z nich poměrně snadno vytvořit strukturu databáze), než diagramy tříd.

Struktura databázového systému byla vytvořena z ER diagramu a nad touto strukturou vytvořeny uložené procedury, pro jednodušší pozdější dotazování. Návrh některých uložených procedur byl poměrně komplikovaný, jelikož databázová struktura byla navržena tak, aby byla v třetí normální formě. To zapříčinilo odstranění redundance, ale na úkor zatížení databáze. Na testované verzi se toto nikterak viditelně neprojevovalo. V případě delší doby zpracování tohoto dotazu, bude muset být pozměněna struktura databáze na úkor nějaké redundance.

Systém se nyní nachází ve fázi testování a následně bude pokračovat vývoj tohoto systému. V dalších fázích bude vývoj zaměřen na vytvoření vylepšeného modulu tisku, kde si může uživatel nadefinovat vlastní vzhled, zobrazování statistik prodejů jednotlivých druhu zboží. V modulu směn bude vytvořena komponenta, která bude přehledněji přiřazovat zaměstnance na směny a případně vytvořeno webové rozhraní pro zobrazení rozpisu směn na internetu.

Aplikace bude používána z počátku ve dvou kavárnách a jednom barovém zařízení. Součástí jedné z kaváren je také výroba zmrzliny, u které se využije také kniha norem.

Literatura

[1] Arlow, J., Neustadt, I. UML2 a unifikovaný proces vývoje aplikací. Brno, Computer Press, a.s. 2007

[2] Maciaszek, A. L., Requirements Analysis And System Design, II., Gosport, Ashford Colour Press Ltd, 2005

[3] Brainstormiong, 26.3.2008. Dokument dostupný na URL http://cs.wikipedia.org/wiki/Brainstorming (prosinec 2007)

[4] Joint Application Development. Dokument dostupný na URL http://cevis.datis.cdrail.cz/MRKP/5/5423JointApplDevelopment.html (prosinec 2007)

[5] Rational Unified Process, IBM. Dokument dostupný na URL http://www-306.ibm.com/software/awdtools/rup/index.html, prosinec 2007

[6] Model zralosti procesů tvorby softwaru, SystemOnLine, 2005. Dokument dostupný na URL http://www.systemonline.cz/clanky/model-zralosti-procesu-tvorby-software.htm (prosinec 2007)

[7] Křena, B., Kočí, R., Úvod do softwarového inženýrství – studijní opora, 2006

[8] Rattz, Joseph, C., Jr., Pro LINQ, APRESS, 2007

[9] The LINQ Project, Microsoft. Dokument dostupný na URL http://msdn.microsoft.com/en-us/netframework/aa904594.aspx (květen 2008)

[10] Úvod do LINQ, Vyvojar.cz, 24.1.2008. Dokument dostupný na URL http://www.vyvojar.cz/Articles/563-uvod-do-linq.aspx, květen 2008

[11] Gross, Christian, Beginning C# 2008 From Novice to Profesional, APRESS, 2007

[12] Agarwal, Vidya Vrat, Beginning C# 2008 Databases From Novice to Profesional, APRESS 2007

[13] Hilyard, Jay, C# 3.0 Cookbook, O'REILLY, 2007

[14] Jaziersky, Edward A., Application Architecture for .NET: Designing Applications and Services, Microsoft Corporation, 2002

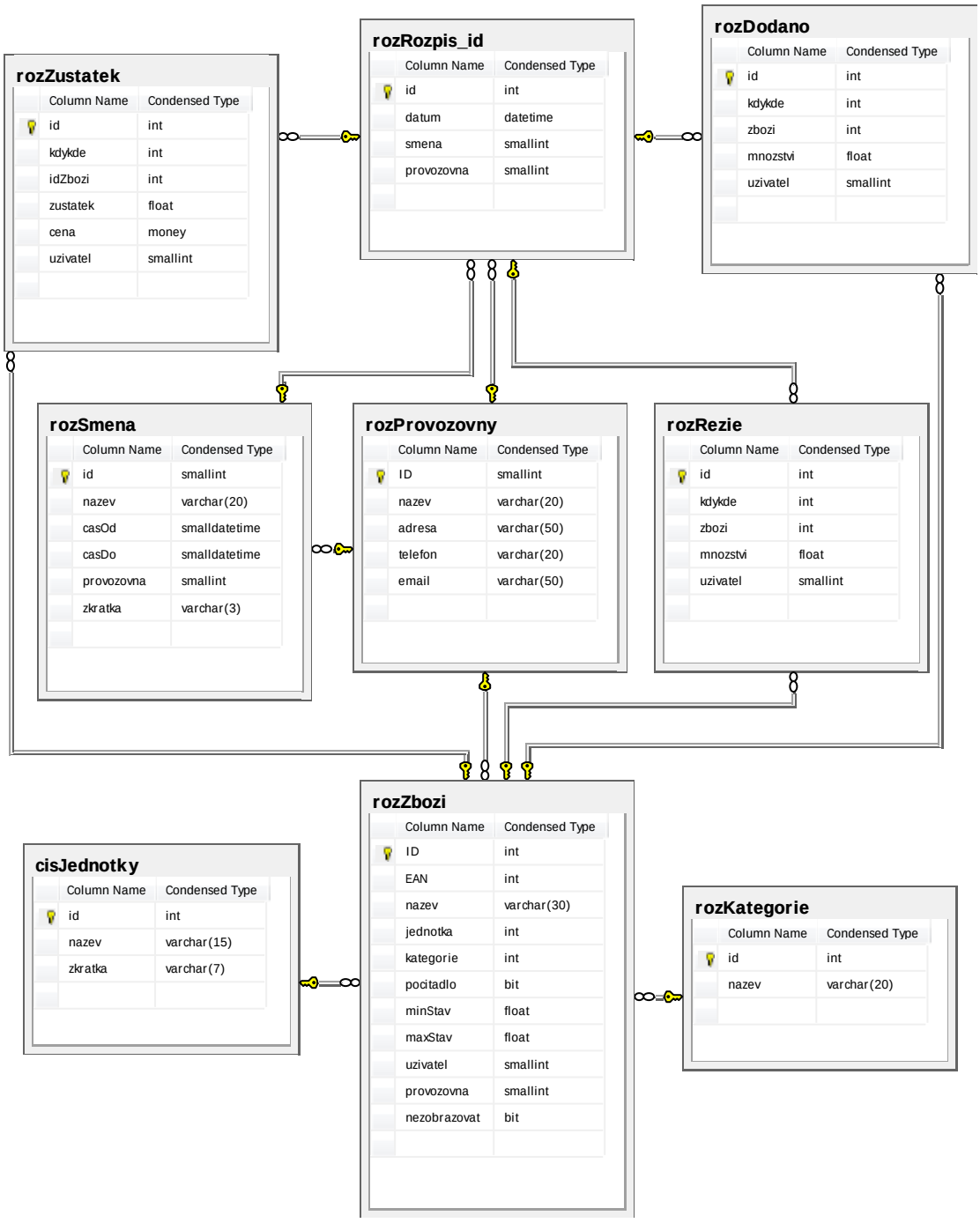
[15] Prosise, J. Programování v Microsoft .NET: webové aplikace v .NET Framework, C# a ASP.NET

[16] Vyvyjíme databázový a informační systém, Databázový svět, 5.5.2004. Dokument dostupný na URL http://www.dbsvet.cz/view.php?cisloclanku=2004050501, duben 2008

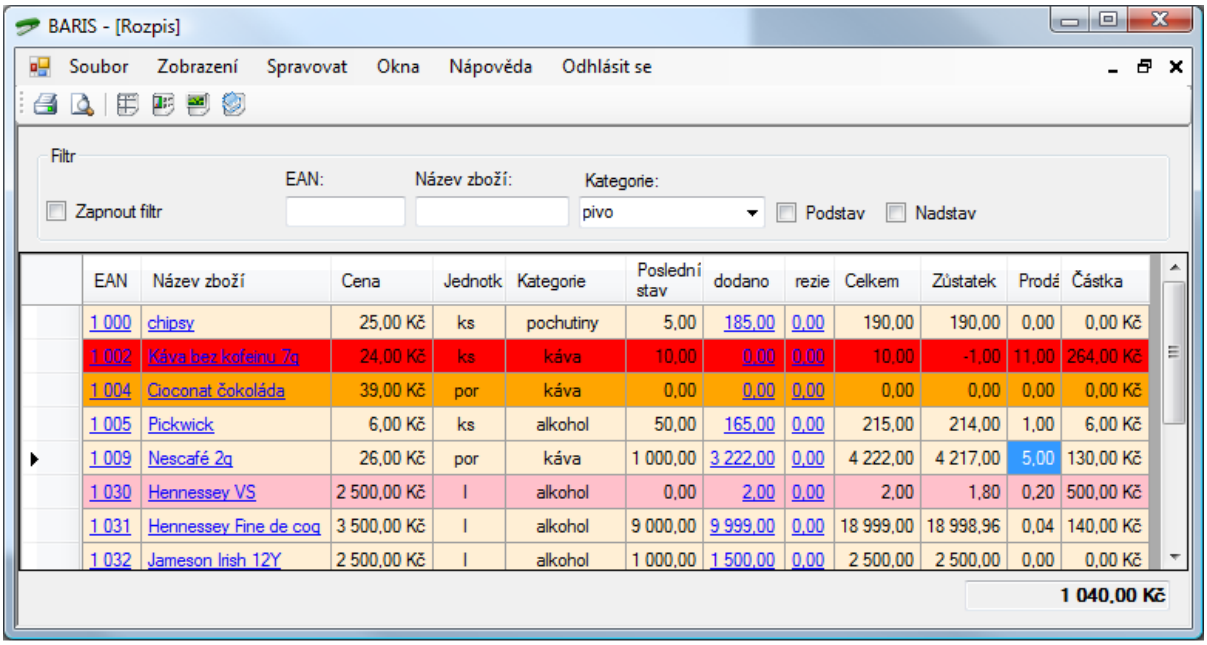
[17] Lacko, L., SQL – hotová řešení, Computer Press, 2003

Příloha

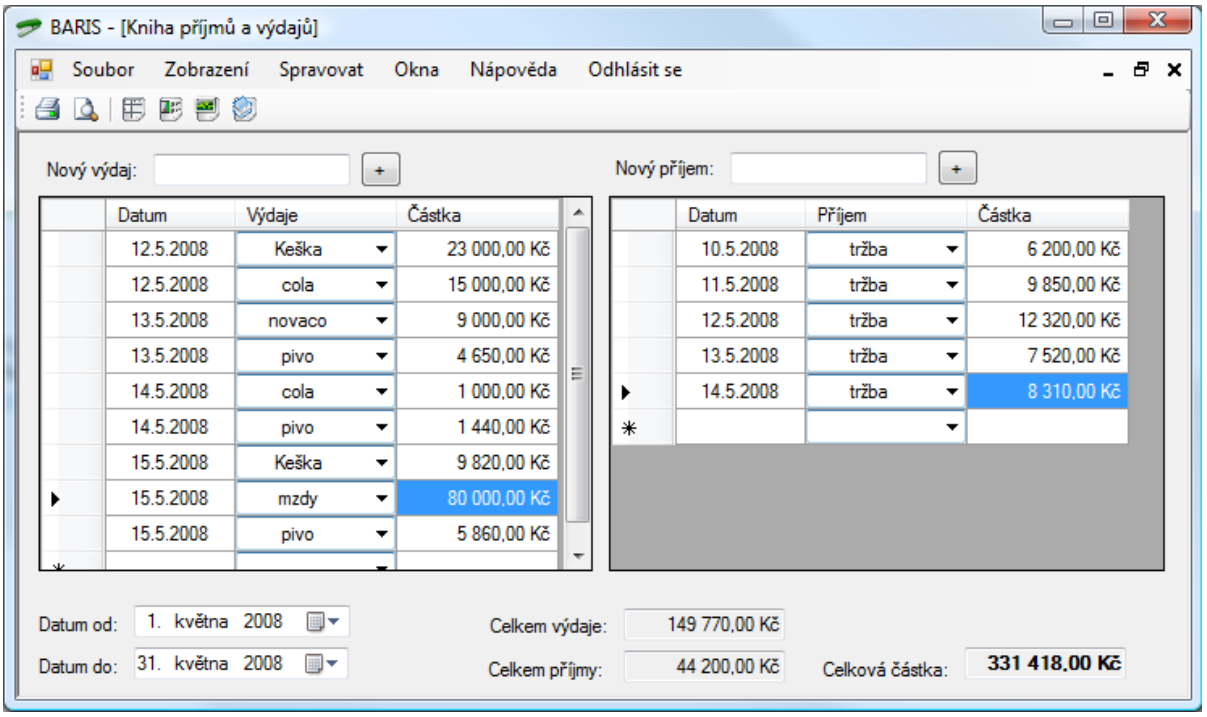
Struktura databáze



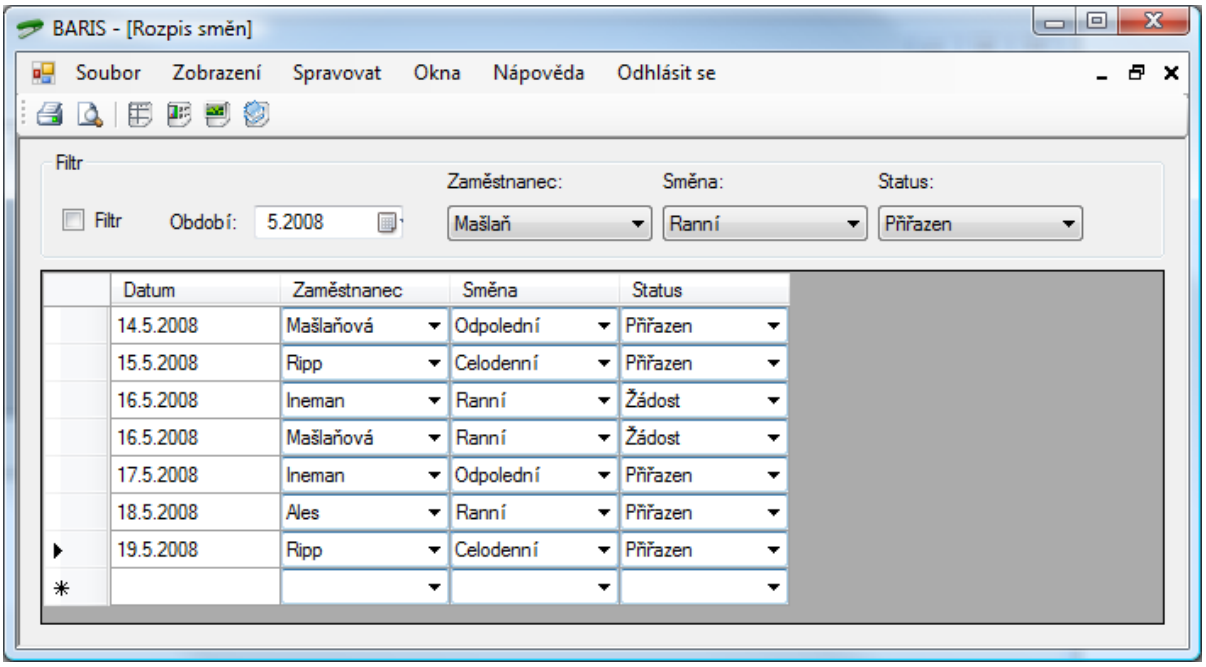
Struktura databáze – Evidence zboží



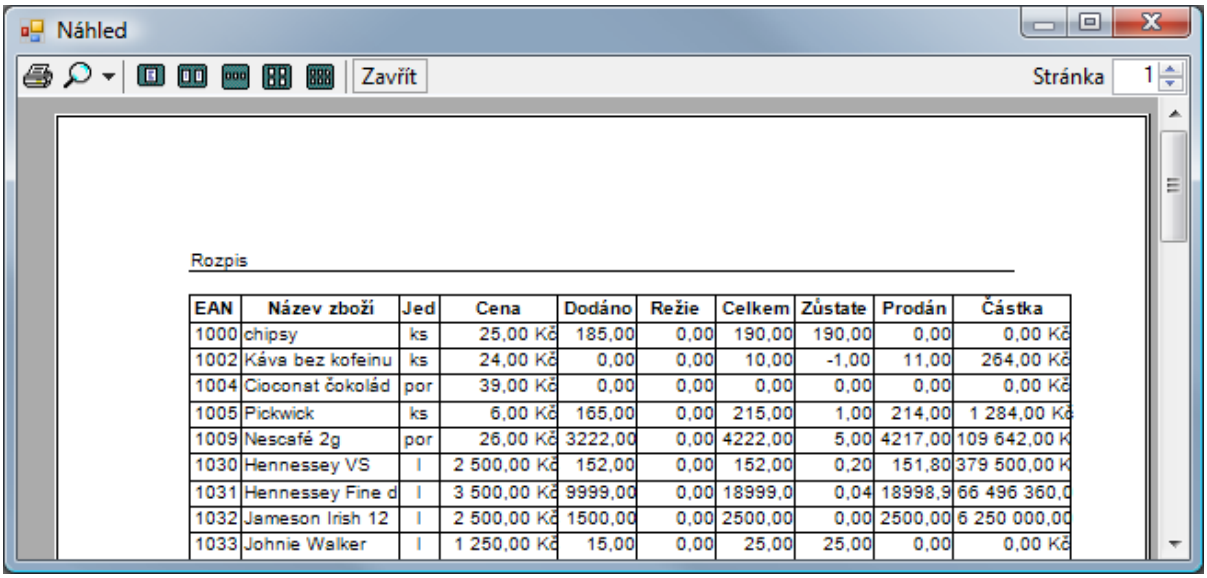
Obrázek aplikace - rozpis



Obrázek aplikace - Kniha příjmů a výdajů



Obrázek aplikace – Rozpis směn



Obrázek aplikace - náhlad tisku