

# MoneyFlow app

Předmět: X33TSW – Testování a kvalita software

Řešitelé:

**Josef Lobotka** / lobotjos@fel.cvut.cz (podíl **50%**), 3. ročník

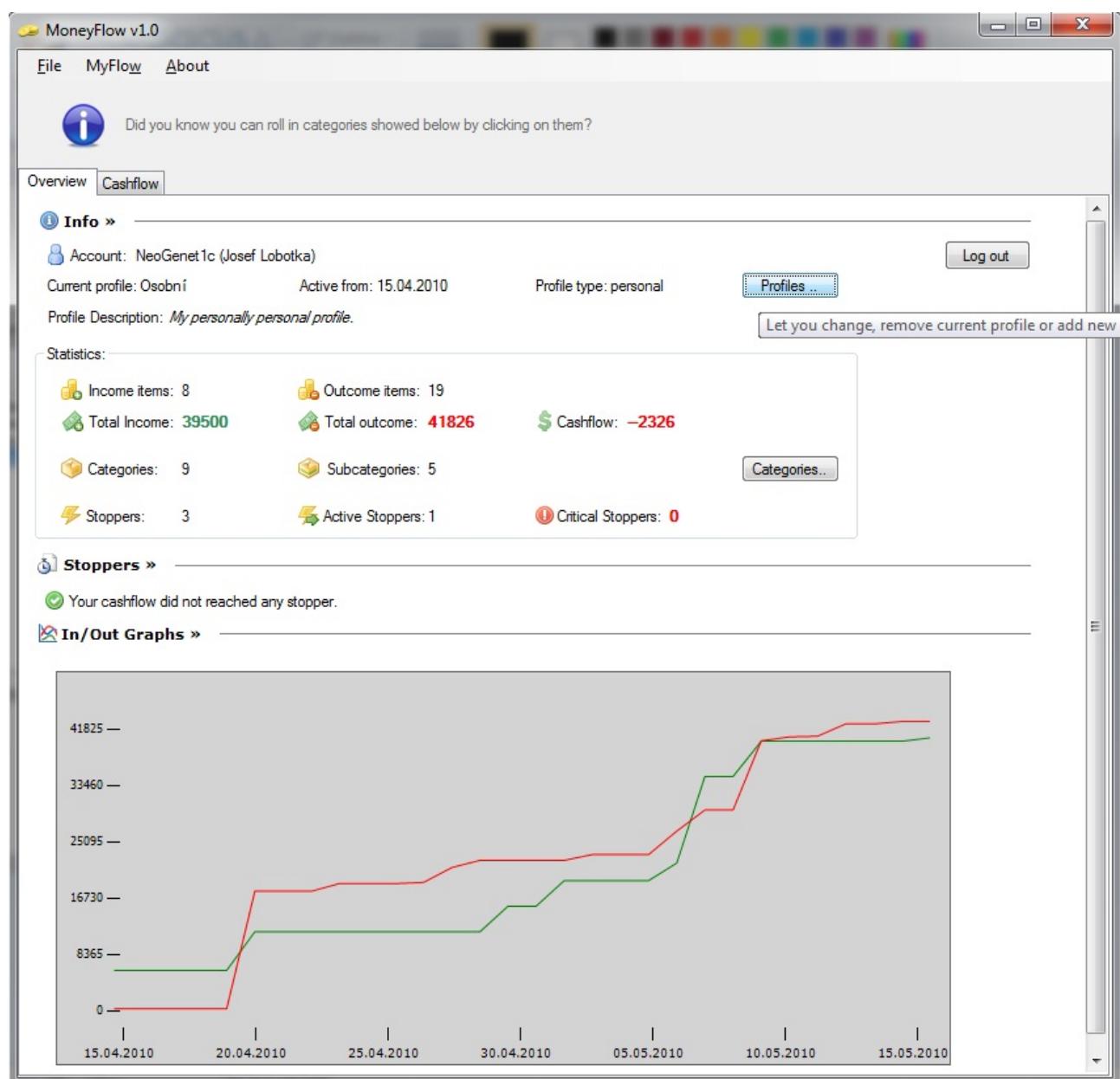
**Martin Lukeš** / wox2@seznam.cz (podíl **50%**), 3. ročník

Datum odevzdání práce: **30.11.2010**

## 1.1 Popis software

Aplikace **MoneyFlow** (obecně „cashflow manager“) má primárně sloužit k rychlému a efektivnímu udržování přehledu o výdeji a příjmu financí uživatele. Kladen je důraz na jednoduchost, snadnou usability a maximální UX tak, aby aplikaci mohli využívat jak začátečníci, tak profesionálové a dokonce majitelé menších a středních podniků.

MoneyFlow je napsaná v jazyku C#, využívá tedy framework .NET ve verzi 3,5 a musí běžet v prostředí Windows 98 a vyšších. Jako databáze byla zvolena velmi jednoduchá SQL Compact, jejíž ovladače je pro používání aplikace potřeba nainstalovat (v budoucnu budou automaticky nainstalovány z instalačního balíčku aplikace). Jako softwarová architektura byla použita třívrstvá MVC (model-view-controller), kdy na datové vrstvě je využíváno microsoftí ORM Linq2SQL pro snadnější portabilitu mezi MS databázemi. Screenshot GUI aplikace je na zachycen na *Obr. 1*.



*Obr. 1 – GUI aplikace MoneyFlow*

## 1.2 Systémové a softwarové požadavky

### 1.2.1.a Hlavní systémové (funkční) požadavky

- REQ00 – **Správa Uživatelů**

Systém umožňuje správu uživatelských účtů, tedy vytvářet uživatele, měnit jejich nastavení a mazat uživatele.

- REQ04 – **Správa Profilů**

Systém umožňuje správu profilů pod každým uživatelem, tedy vytvářet profily, rozlišovat je, měnit a mazat.

- REQ08 – **Správa Cashflow**

Systém umožňuje správu příjmů a výdajů a úkonů s tím spojených, tedy vytváření, změny a mazání kategorií a subkategorií v jednotlivých profilech a přidávání, změny a mazání příjmových / výdajových položek.

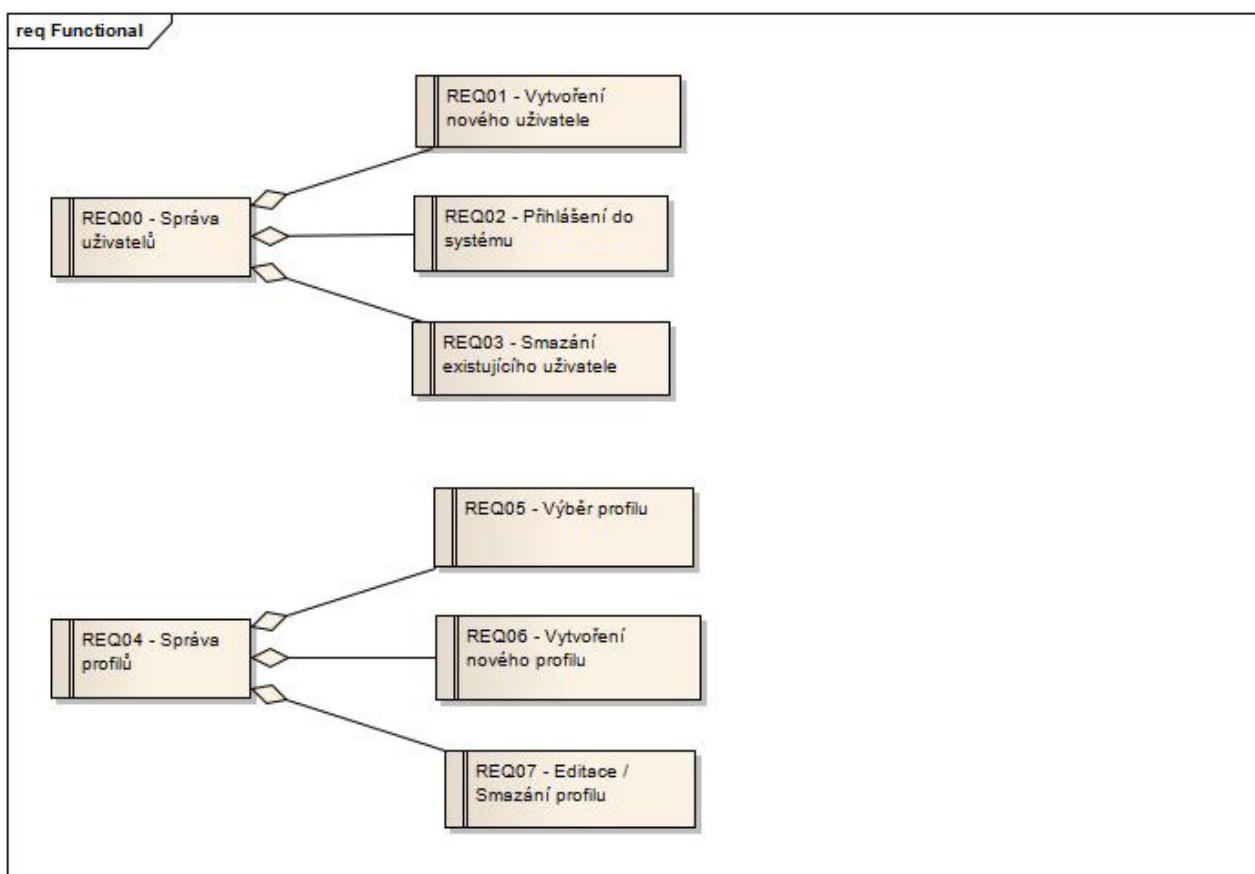
- REQ20 – **Analýza Cashflow**

Systém umožňuje zobrazení analytických informací o vedených položkách cashflow daného profilu.

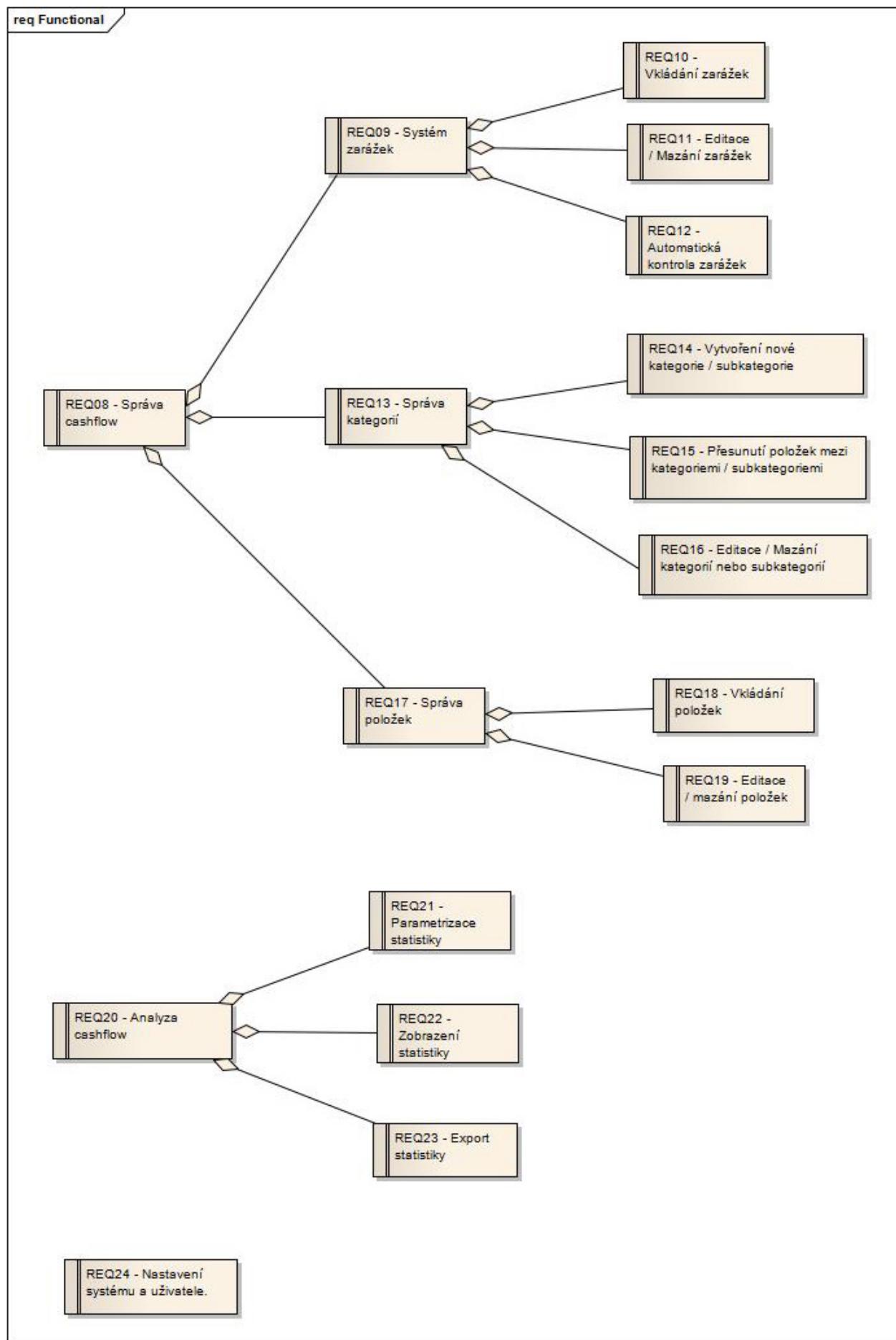
- REQ24 – **Nastavení uživatele**

Systém bude umožňovat měnit informace o uživateli a nastavení uživatele.

Na obrázku 2 a obrázku 3 jsou zachyceny závislosti mezi softwarovými požadavky.



Obr. 2 – Závislosti mezi softwarovými požadavky, 1. část



Obr. 3 – Závislosti mezi softwarovými požadavky, 2. část

### **1.2.1.b Hlavní softwarové (nefunkční) požadavky**

- **REQ25 – Portabilita mezi počítači**  
Systém je přenosný mezi různými počítači (splňujícími minimální požadavky na běh aplikace) a to včetně již vytvořených dat.
- **REQ26 – Jednoduché GUI rozhraní**  
Systém poskytuje jednoduché, intuitivní a uživatelsky přívětivé grafické rozhraní.
- **REQ27 – Operační systém Windows**  
Systém musí běžet pod OS Windows verze 98 a vyšší.
- **REQ28 - .NET Framework**  
Systém musí běžet pod nainstalovaný frameworkm .NET ve verzi 3,5.
- **REQ29 – Ochrana a šifrování dat uživatelů**  
Systém uživatelů umožňuje chránit svá data heslem. Ta jsou pak pro větší bezpečnost šifrována.
- **REQ30 - Nutnost nenáročné konfigurace PC**  
Systém je možné spustit i na velmi slabé PC konfiguraci (stačí podpora OS a .NET frameworku 3,5).

### **1.2.2 Use Cases (případy užití)**

Name: **Vytvořit nového uživatele**

Identifier: **UC01**

Description: Popis vytvoření nového uživatelské účtu aplikace.

Actors: Common user

Preconditions:

- Aplikace ihned po spuštění NEBO pokud se dosavadní uživatel úspěšně odhlásil.

Postconditions:

- Systém úspěšně vytvořil nového uživatele a informoval ho o této akci.
- Vytvořený uživatel je vidět ve výpisu všech uživatelů a může se nyní přihlásit do systému.

Basic course of action:

1. UC začíná kliknutím na tlačítko "Založit nový účet".
2. Systém zobrazí registrační formulář.
3. Uživatel vyplní základní údaje, konkrétně: Jméno, příjmení, login, heslo (2x - pro kontrolu správného zadání hesla) a potvrdí formulář tlačítkem "Založit účet". [Extension point: **UC04 Proveďte validaci vložených údajů**]
4. V případě úspěšné validace pokračuje systém bodem 5, jinak vypíše příslušnou chybovou hlášku a vrací se k bodu 3.
5. Systém zruší registrační formulář, vrátí uživatele na předchozí okno a vypíše "Byl vytvořen nový zákazník + login\_zakaznika".

**Name: Přihlásit se do systému****Identifier: UC02**

Description: Popis přihlášení uživatele do systému.

Actors: Common user

Preconditions:

- V systému musí již být vytvořen minimálně jeden uživatel (viz UC01)

Postconditions:

- Systém uživatele informoval o úspěšném přihlášení.
- Systém uživatele přenesl z přihlašovací obrazovky do hlavního prostředí aplikace.

Basic course of action:

1. UC začíná, když uživatel spustí aplikaci..
2. Systém zobrazí uvítací okno s přihlašovacím formulářem. Uvítací okno bude obsahovat text, logo apod., přihlašovací formulář pak seznam již zřízených loginů, heslo, tlačítko "Přihlásit se" a tlačítko "Vytvořit nového uživatele" (UC01).
3. Uživatel vybere login a vyplní heslo. [Extension point: UC04 Provede validaci vložených údajů]
4. V případě, že proběhne validace v pořadku UC končí, jinak se vrací na krok 4 a vypíše chybovou hlášku "Nesprávné heslo".

**Name: Smazat existujícího uživatele****Identifier: UC03**

Description: Popis postupu pro smazání již existujícího uživatele.

Actors: Common user

Preconditions:

- V systému musí již být vytvořen minimálně jeden uživatel (viz UC01)

Postconditions:

- Systém uživatele informoval o úspěšném smazání uživatelského konta.
- Systém uživatele odebral ze seznamu uživatelů.

Basic course of action:

1. UC začíná, když uživatel stiskne tlačítko "Smazat účet".
2. Systém uživatele vyzve k zadání svého hesla.
3. Uživatel zadá heslo.
4. Systém ověří, zda-li se heslo shoduje s heslem k danému účtu, pokud ne, vypíše chybovou hlášku "Nesprávné heslo" a vrátí se k bodu 2. Pokud ano, pokračuje bodem 5.
5. Systém se uživatele zeptá, jestli si je smazáním celého účtu opravdu jistý.
6. Pokud uživatel stiskne tlačítko "Definitivně smazat účet", dojde ke smazání účtu. Pokud stiskne "Nemazat", nedojde k jeho smazání.
7. Systém vypíše příslušnou hlášku a vrátí uživatele na předchozí obrazovku.

Name: **Validace vložených údajů**

Identifier: **UC04**

Description: Popis postupu pro ověření uživatelem vložených údajů

Actors: Common user

Preconditions:

- Uživatel musí odeslat formulář, který se má validovat.

Postconditions:

- Systém definitivně určí jestli validace proběhla úspěšně nebo neúspěšně.

Basic course of action:

1. UC začíná, když je systém vyzván k ověření údajů vložených do formuláře.
2. Systém provede kontrolu vložených údajů (např. jestli odpovídá délka, jestli je vložený údaj opravdu e-mail atd.).
3. Systém vyhodnotí jestli jsou vložené údaje správné nebo nesprávné a tímto UC končí.

### **1.2.3.a Traceability Matrix**

Vyjádření vztahů systémových (funkčních) a softwarových (nefunkčních) požadavků zachycených pomocí traceability matrix na obrázku č. 4.

	Non-Functional::REQ25 - Portabilita mezi zařízení	Non-Functional::REQ26 - Jednoduché GUI rozhraní	Non-Functional::REQ27 - Operační systém Windows	Non-Functional::REQ28 - .NET Framework	Non-Functional::REQ29 - Ochrana a šifrování dat uživatelů	Non-Functional::REQ30 - REQ30 - Nutnost nenáročnosti
Functional::REQ00 - Správa uživatelů						
Functional::REQ01 - Vytvoření nového uživatele						
Functional::REQ02 - Přihlášení do systému						
Functional::REQ03 - Smazání existujícího uživatele						
Functional::REQ04 - Správa profilů						
Functional::REQ05 - Výběr profilu						
Functional::REQ06 - Vytvoření nového profilu						
Functional::REQ07 - Editace / Smazání profilu						
Functional::REQ08 - Správa cashflow						
Functional::REQ09 - Systém zarážek						
Functional::REQ10 - Vkládání zarážek						
Functional::REQ11 - Editace / Mazání zarážek						
Functional::REQ12 - Automatická kontrola zarážek						
Functional::REQ13 - Správa kategorií						
Functional::REQ14 - Vytvoření nové kategorie / subkateg...						
Functional::REQ15 - Přesunutí položek mezi kategoriemi...						
Functional::REQ16 - Editace / Mazání kategorií nebo su...						
Functional::REQ17 - Správa položek						
Functional::REQ18 - Vkládání položek						

#### Obr. 4 – Traceability matrix

### 1.2.3.b Attributes Matrix

Matrice atributů zobrazuje status, složitost, prioritu, typ, datum poslední editace a datum vytvoření daných požadavků. Data pocházejí z programu Enterprise Architect viz *tabulky 1 a 2*.

Ze stavu požadavků týkajících se zarážek a statistik se dá vydedukovat, že aplikace není kompletní, a jejich implementace bude doplněna v příštích iteracích.

Requirements	Status	Difficulty	Priority	Type	Last Update	Created
REQ00 Správa uživatelů	implemented	medium	high	functional	2.10.2010	18.4.2010
REQ01 Vytvoření nového uživatele	implemented	medium	high	functional	2.10.2010	19.4.2010
REQ02 Přihlášení do systému	implemented	medium	high	functional	2.10.2010	20.4.2010
REQ03 Smazání existujícího uživatele	approved	medium	high	functional	21.4.2010	21.4.2010
REQ05 Výběr profilu	implemented	medium	high	functional	2.10.2010	22.4.2010
REQ06 Vytvoření nového profilu	implemented	medium	high	functional	2.10.2010	23.4.2010
REQ07 Editace/Smazání profilu	implemented	medium	high	functional	2.10.2010	24.4.2010
REQ08 Správa Cashflow	implemented	medium	high	functional	2.10.2010	25.4.2010
REQ09 Systém zarážek	proposed	medium	low	functional/display	18.4.2010	26.4.2010
REQ10 Vkládání zarážek	proposed	medium	low	functional	18.4.2010	27.4.2010
REQ11 Editace/Mazání zarážek	proposed	medium	low	functional	18.4.2010	28.4.2010
REQ12 Automatická kontrola Zarážek	proposed	high	low	validate	18.4.2010	29.4.2010
REQ13 Správa kategorií	implemented	low	medium	functional	2.10.2010	30.4.2010
REQ14 Vytvoření nové kategorie/subkategorií	implemented	low	medium	functional	2.10.2010	1.5.2010
REQ15 Přesunutí položek mezi kategoriemi	implemented	low	medium	functional	2.10.2010	2.5.2010
REQ16 Editace / Mazání kategorií nebo subkategorií	implemented	low	medium	functional	2.10.2010	3.5.2010

Tab 1. - Attribute Matrix 1. část

Requirements	Status	Difficulty	Priority	Type	Last Update	Created
REQ17 Správa položek	implemented	low	high	functional	2.10.2010	18.4.2010
REQ18 Vkládání položek	implemented	low	high	functional	2.10.2010	18.4.2010
REQ19 Editace položek	implemented	low	high	functional	2.10.2010	18.4.2010
REQ25 Mazání položek	implemented	low	high	functional	2.10.2010	18.4.2010
REQ20 Analýza cashflow	proposed	high	high	functional	18.4.2010	18.4.2010
REQ21 Parametrisace cashflow	proposed	high	medium	functional	18.4.2010	18.4.2010
REQ22 Zobrazení statistiky	proposed	high	medium	display	18.4.2010	18.4.2010
REQ23 Export Statistiky	proposed	high	medium	functional	18.4.2010	18.4.2010

Tab 2. - Attribute Matrix 2. část

#### 1.2.4 Test cases

Name: **Vytvořit nového uživatele**

Identifier: **TC01**

Description: Test Case ověřující správnou implementaci UC01 Vytvoření nového uživatelem

Step	Procedure	Success Criteria
1	klikni na tlačítko Create new account	formulář pro vyplnění přihlašovacích údajů zobrazen
2	zadej nick a zmáčkní tlačítko Create new user account	Zobrazení chybové hlášky + žádosti o zadání hesla
3	zadej krátké heslo (méně než šest znaků) do oken Password a Repeat Password a stiskni Create new user account	Zobrazení chybové hlášky + žádosti o zadání nejméně 6 znakového hesla
4	zadej dlouhé heslo (6 a více znaků) do kolonky Password a stiskni Create new user account	Zobrazení chybové hlášky + žádosti o korektní zadání hesla a potvrzení hesla
5	zadej dlouhé heslo (6 a více znaků) do kolonky Password a Repeat Password, stiskni Create new user account	Zobrazení logovacího menu s zprávou o vytvoření nového účtu

Name: **Přihlásit se do systému**

Identifier: **TC02**

Description: Test Case ověřující správnou implementaci UC02 Přihlásit se do systému

Step	Procedure	Success Criteria
1	Vyber účet s známým heslem	Text area pro zadání hesla zpřístupněna
2	Zadej heslo, které se neshoduje s heslem k vybranému účtu	Zobrazení chybové hlášky + žádosti o nové zadání hesla
3	Zadej známé heslo k účtu	Vstup do systému

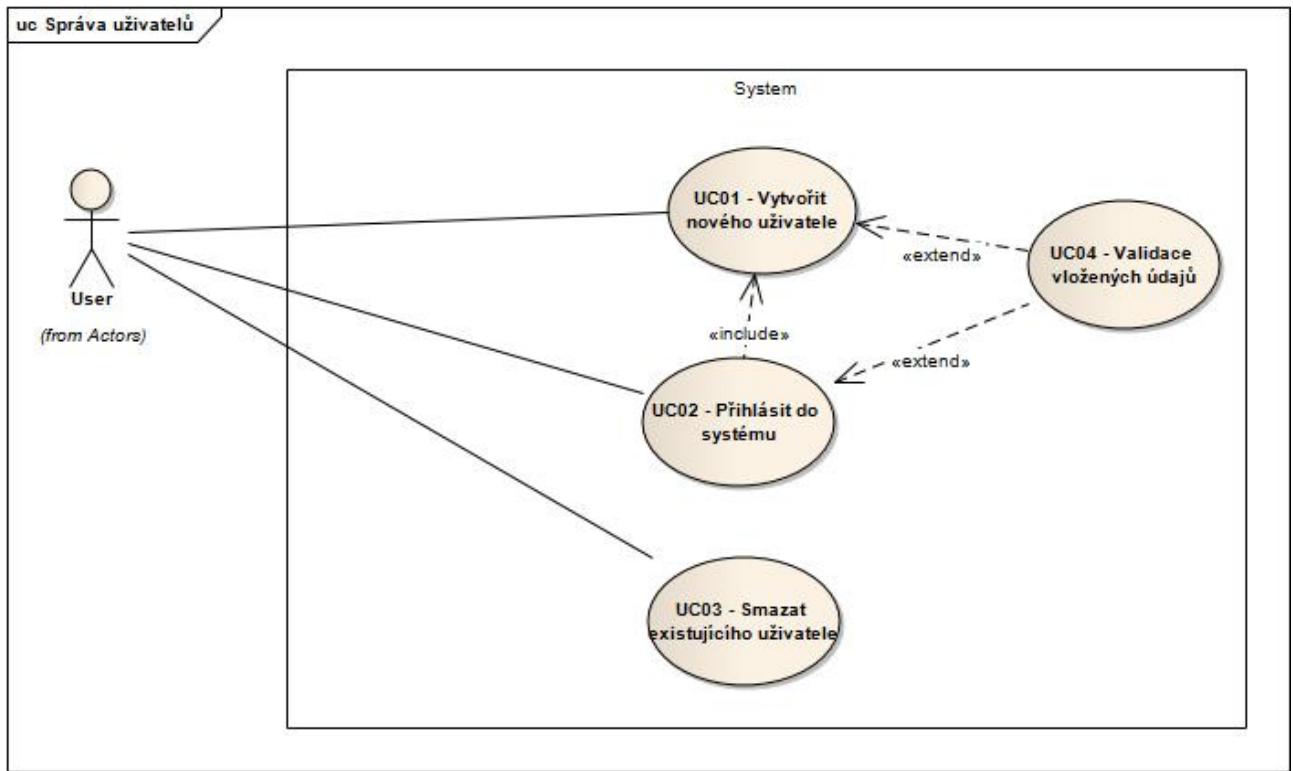
Name: **Editace položek**

Identifier: **TC19**

Description: Test Case ověřující správnou implementaci UC03 Editace položek

## 2.1 Use Case diagram

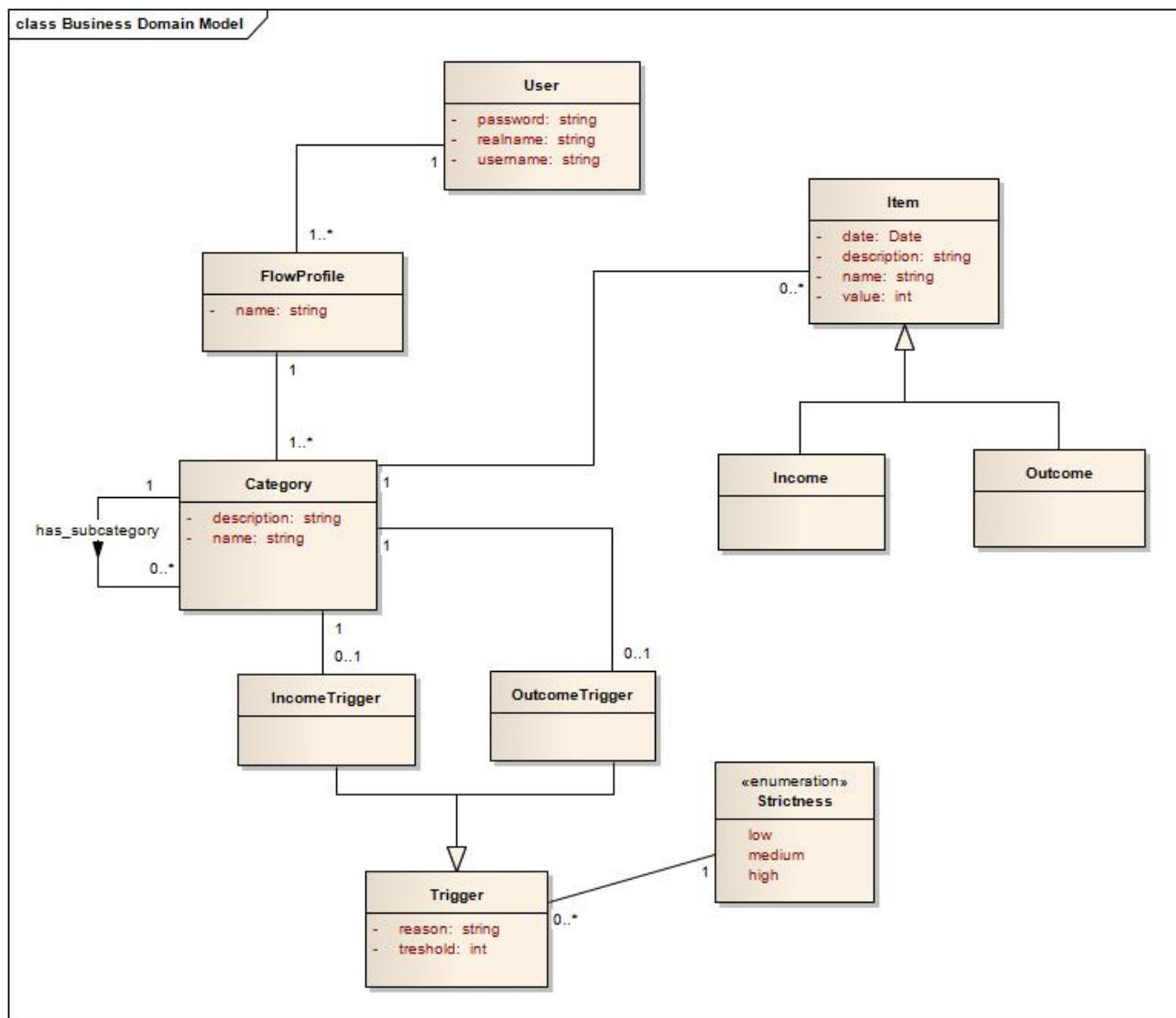
Use Case diagram graficky zachycující případy užití Správy uživatelů popsané v bodě 2.2 je zobrazen na *Obrázku 5*.



*Obr. 5 – Diagram případů užití Správy uživatelů*

## 2.2 Class diagram

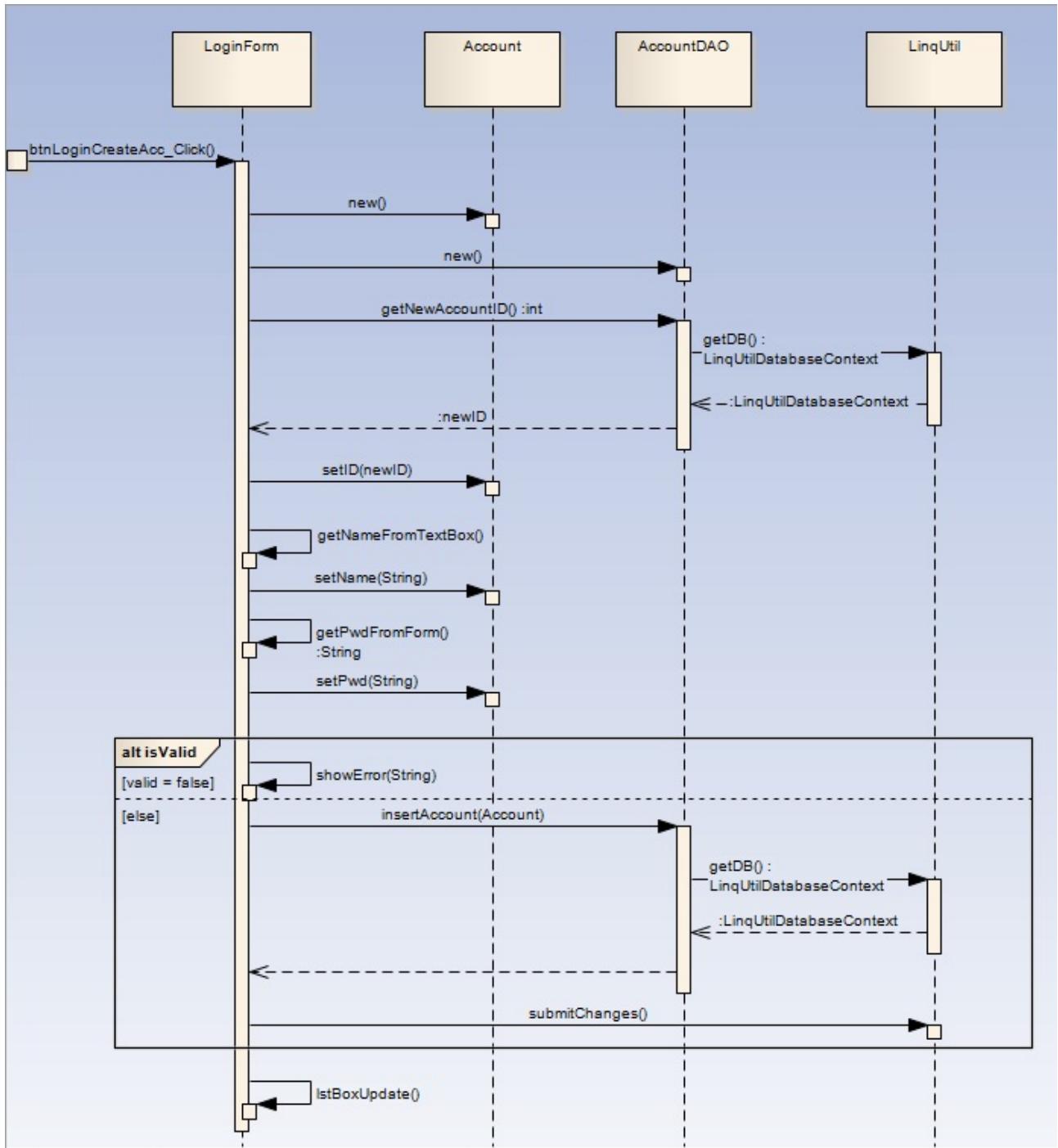
K lepšímu pochopení struktury projektu zobrazení závislostí mezi jednotlivými třídami jsme si vytvořili class diagram projektu, který je zobrazen na Obrázku 6.



Obr. 6 – Diagram použitých tříd v projektu (class diagram)

## 2.3 Sequence diagram

Sekvenční diagram případu užití Vytvořit nového uživatele (UC01) zobrazený na Obrázku 7 nám přibližuje komunikaci mezi třídami LoginForm, Account, AccountDAO a knihovní třídou LinqUtil. Spolupráce mezi účastníky komunikace je typickým příkladem distribuovaného řízení.



Obr. 7 – Sekvenční diagram průběhu vytvoření nového uživatele

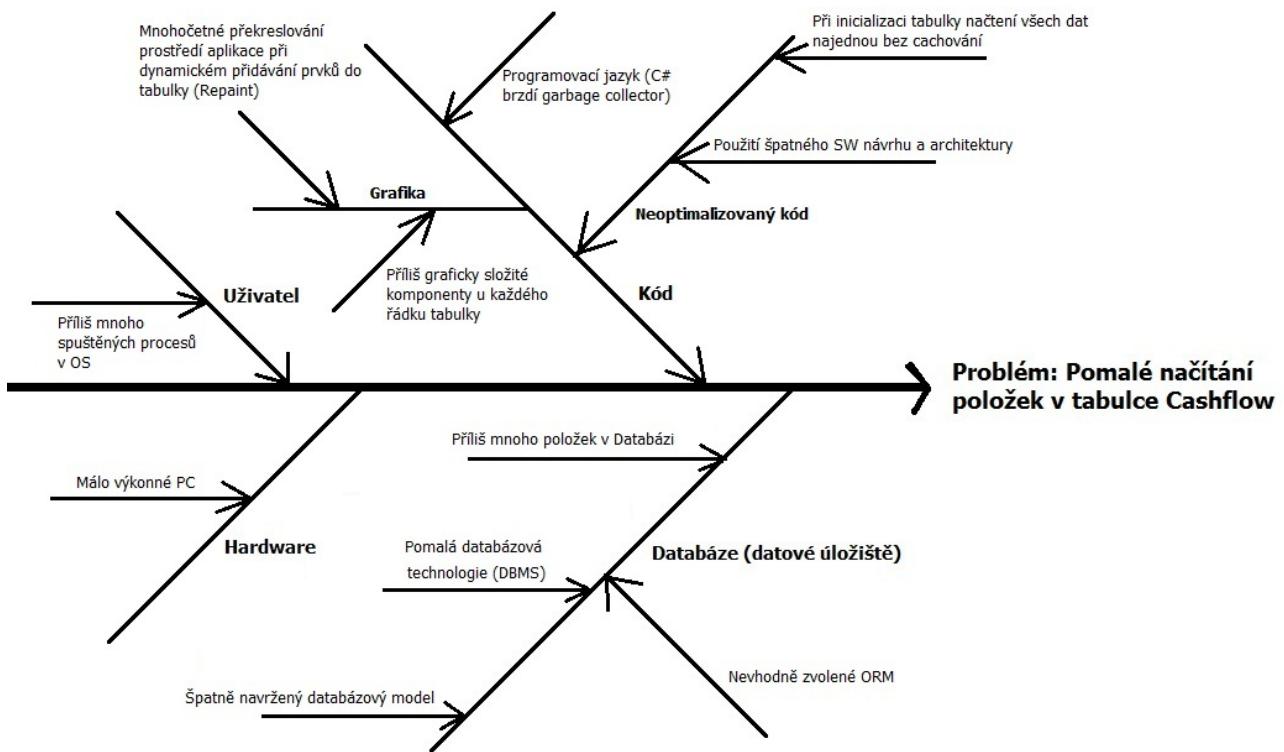
### **Popis:**

1. Řízení událostí vstupuje na diagram z neurčitého místa voláním metody btnLoginCreateAccount(). Toto volání signalizuje, že uživatel klikl na tlačítko Create Account.
2. Účastník komunikace LoginForm posílá zprávu new () - "vytvoření nové instance" - účastníkovi Account.
3. Účastník LoginForm posílá zprávu new () - "vytvoření nové instance" - účastníkovi AccountDAO.
4. Účastník LoginForm posílá zprávu getNewAccountId () účastníku AccountDao, který získává LinqUtilDatabaseContext od účastníka LinqUtil a následně vrací newId LoginFormu.
5. LoginForm posílá zprávu setId(newID) účastníkovi Account.
6. LoginForm posílá zprávu getNameFromTextBox sám sobě.
7. LoginForm posílá zprávu setName(String) účastníkovi Account.
8. LoginForm posílá zprávu getPwdFromForm() sám sobě a získává zpět návratovou hodnotu typu String.
9. LoginForm posílá zprávu setPwd(String) účastníkovi Account.
10. V Boundary Alt jsou dvě možné cesty určené hodnotou proměnné valid:
  - hodnota valid je true – LoginForm pošle zpávu ShowError(String) sám sobě.
  - v zbylých případech je poslána zpráva insertAccount(Account) účastníkovi AccountDao, který voláním getDB() získává LinqUtilContext od účastníka LinqUtil a vrací řízení LoginFormu. Nakonec LoginForm posílá zprávu submitChanges účastníku LinqUtil.
11. Účastník LoginForm posílá zprávu listBoxUpdate() sám sobě.

### 3.1 CE (Cause & Effect) diagram

Zakreslením a prostudováním CE diagramu (Obrázek 8) jsme rozdělili možné příčiny vzniku problému „Pomalé načítání položek v tabulce Cashflow“ na tyto konkrétní příčiny:

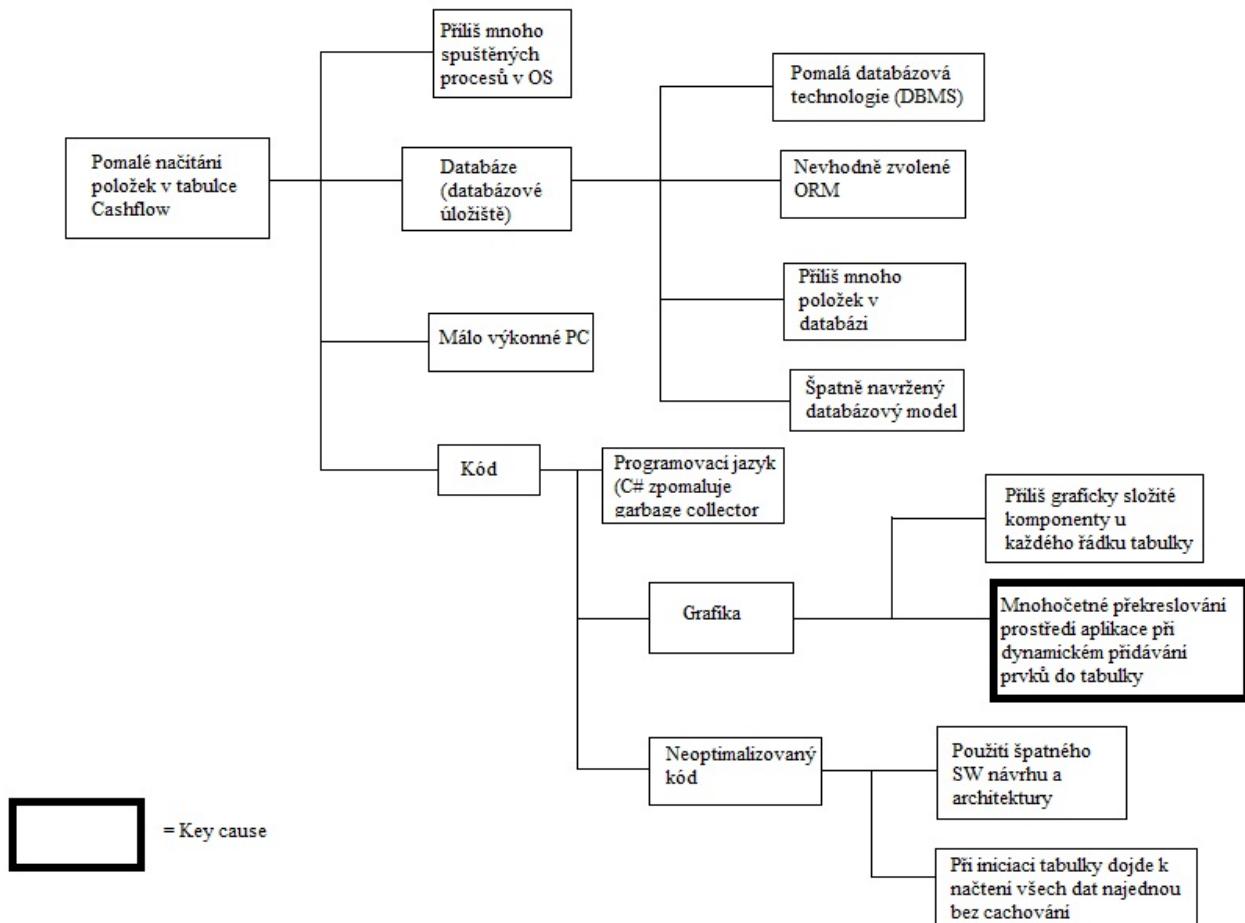
- **Těžko odstranitelné až neodstranitelné**
  1. málo výkonné hardware (PC)
  2. programovací jazyk
  3. mnoho spuštěných procesů v OS – příčina způsobená chováním uživatele
  4. příliš mnoho položek v databázi
- **Příčiny, jež by bylo možné odstranit a tím zamezit vzniku problému**
  1. použití špatného SW návrhu a architektury
  2. načtení všech dat najednou bez cacheování při inicializaci tabulky
  3. mnohočetné překreslování
  4. příliš graficky složité komponenty
  5. pomalá databázová technologie
  6. nevhodně zvolené ORM
  7. špatně navržený databázový model



Obr. 8 – Cause & Effect diagram (CE diagram)

## 3.2 Why why diagram

Pomocí why - why diagramu zobrazujeme stejné skutečnosti jako v CE diagramu, nicméně jsme si navíc označili klíčovou příčinu problému pomalého načítání položek v tabulce Cashflow – **Mnohočetné překreslování prostředí aplikace** (*Obrázek 9*). V diagramu se nám neobjevily duplicitní důvody, běžně označované hvězdičkou (\*).

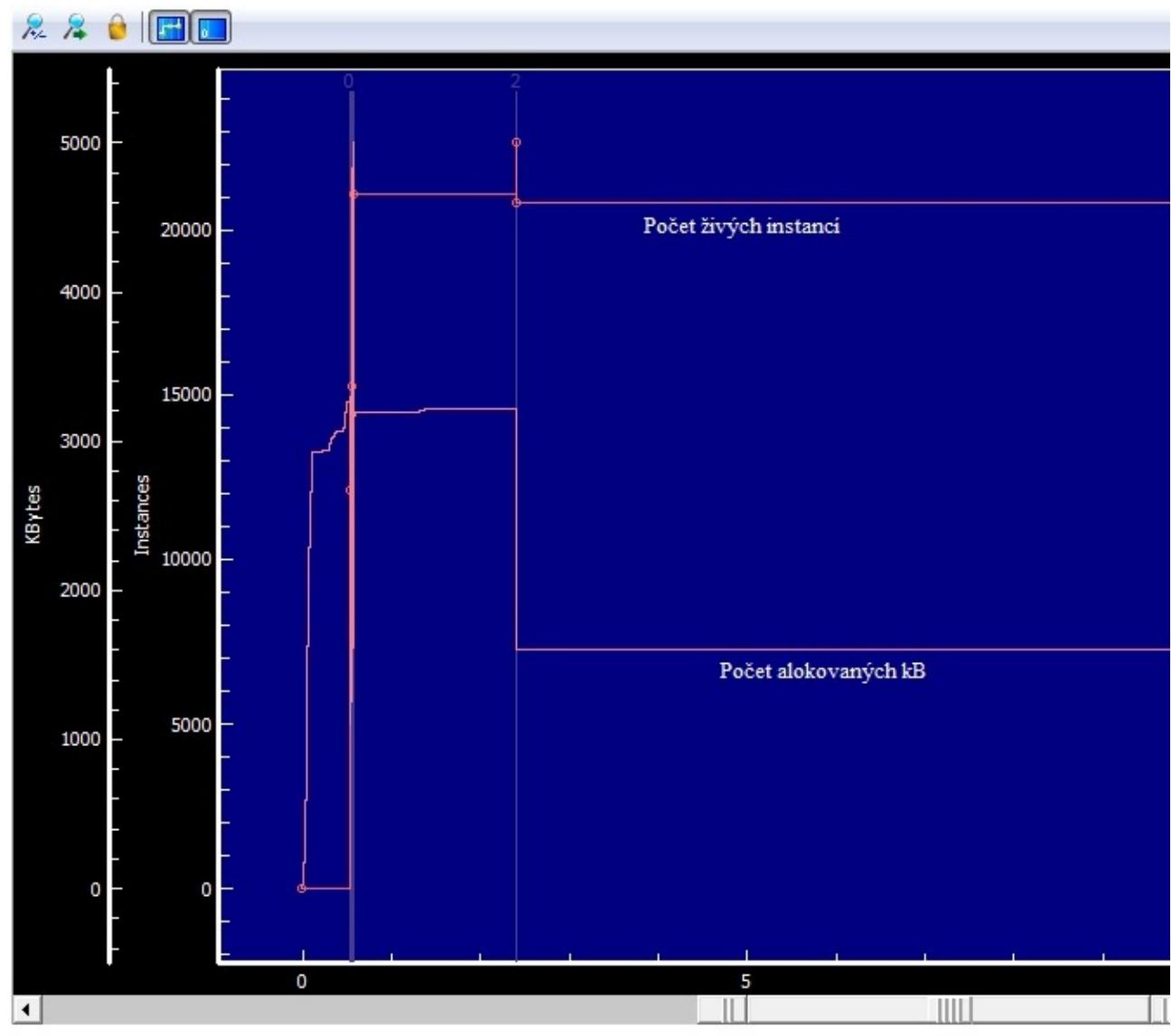


*Obr. 9 – Why-why diagram*

## 4.1 Memory corruption a memory profiling

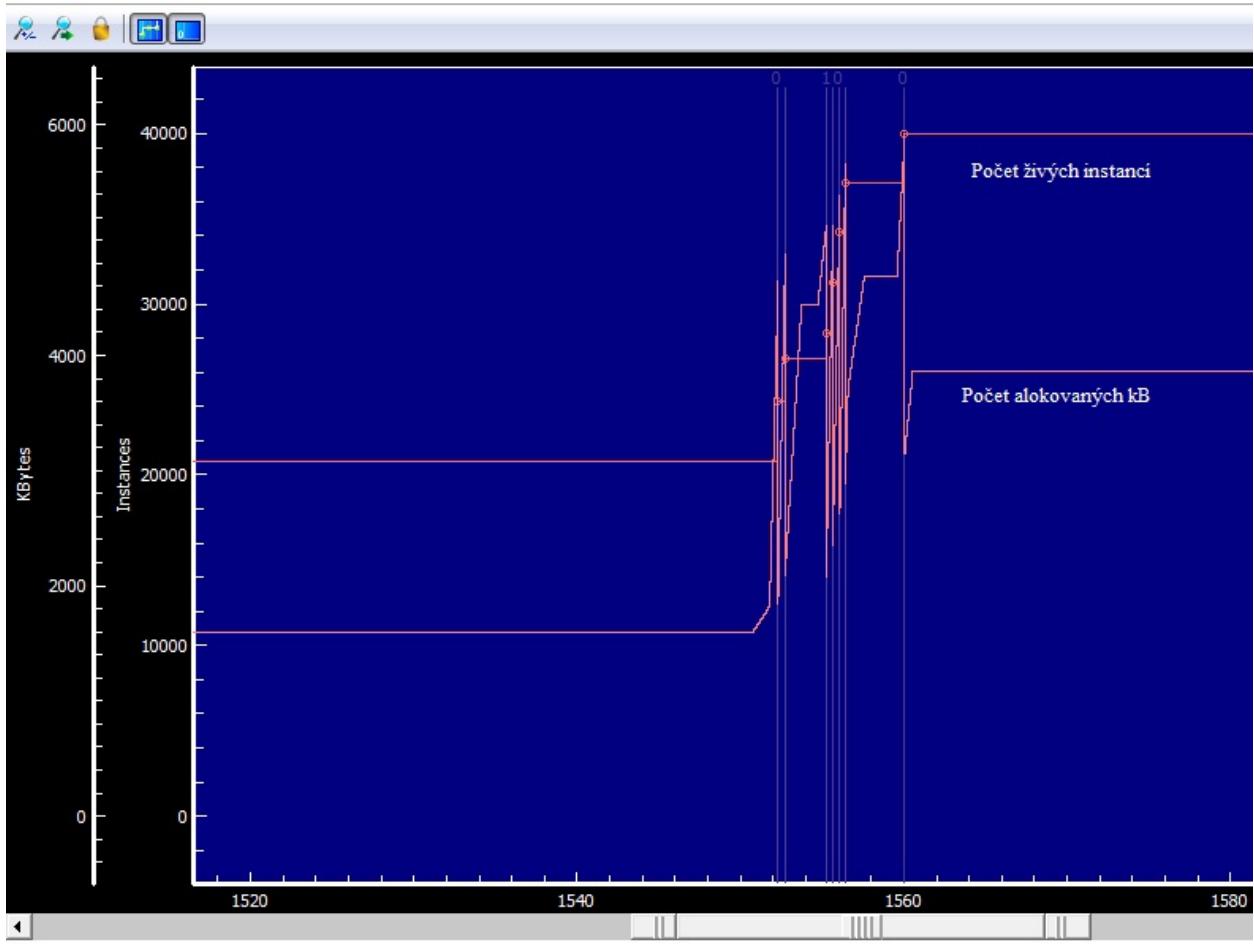
Vzhledem ke skutečnosti, že je naše aplikace napsána v programovacím jazyce C# (s využitím .NET 3.5), se o správu paměti stará velmi efektivní Garbage Collector a tak téměř nedochází k únikům paměti (Memory leakům). Z tohoto důvodu jsme se rozhodli aplikaci otestovat programem .NET Memory profiler 3.5 a určit tak její obecné nedostatky.

Po spuštění a přihlášení uživatele do aplikace nedochází k samovolnému navýšování počtu instancí nebo novým alokacím v paměti (viz *Obrázek 10*), pokud uživatel aplikaci nechá v chodu a nepracuje s ní.



*Obr. 10 - Klidový stav aplikace*

Ve chvíli, kdy uživatel aplikaci začne používat – např. pohne s oknem aplikace, minimalizuje ji, scrolluje, otevřá další dialogy apod. - dochází k rapidním nárustum počtu instancí i alokovaných kB (*Obrázek 11*), ačkoliv nejsou data modifikována a aplikace nemá důvod alokovat více paměti. Tato skutečnost vyvolává doměnu, že je paměť využívána velmi neefektivně.



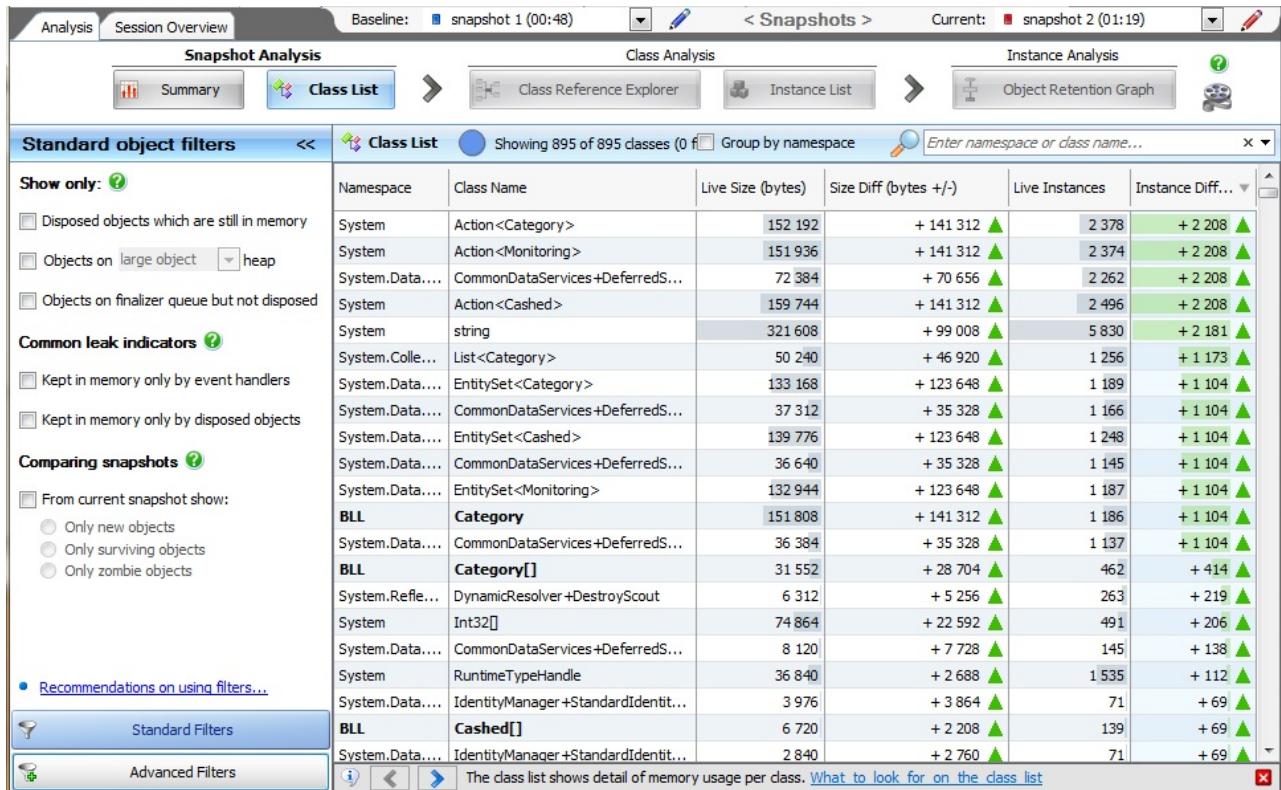
*Obr. 11* - Uživatel začal používat aplikaci

Dalším naším krokem tedy bylo určit které konkrétní instance jsou v aplikaci takto vadně alokovány. Protože .NET Profiler neumí efektivně zobrazovat nárust počtu instancí, pro tento krok jsme použili program Ants Memory Profiler 6.0, ve kterém je možné zobrazit odchylinky mezi dvěma zvolenými snapshoty při průběhu aplikace (*Obrázek 12*). První snapshot byl pořízen ve chvíli, kdy byla aplikace v nečinnosti, druhý snapshot zachycuje stav poté, co uživatel aplikaci chvíli používal.

Z obrázku 12 je velmi dobře patrné, které instance nejvíce alokují nový prostor, ačkoliv uživatel nemění žádné vnitřní objekty. Zvýrazněné třídy jsou pak ty, které jsou součástí naší aplikace, zbytek (nezvýrazněný) jsou třídy C# a frameworku .NET. **Nejvíce instancí tedy vytváří naše třída Category.**

Díky výsledkům našich nálezů jsme ve zdrojovém kódu hlavního řízení aplikace provedli kontrolu všech metod, které jsou využívány pro překreslení okna. V těchto metodách jsme pátrali hlavně po nevhodné manipulaci s instancemi třídy Category. Výsledkem naší kontroly byla velmi neefektivní práce s modelem kategorií, který se při každém překreslování znovunačetl z databáze a vkládal do kolekce, ze které po dokončení překreslení grafu nebyl odstraněn. Právě kvůli tomuto problému docházelo při každém překreslení k vytvoření a uložení nového modelu kategorií, který

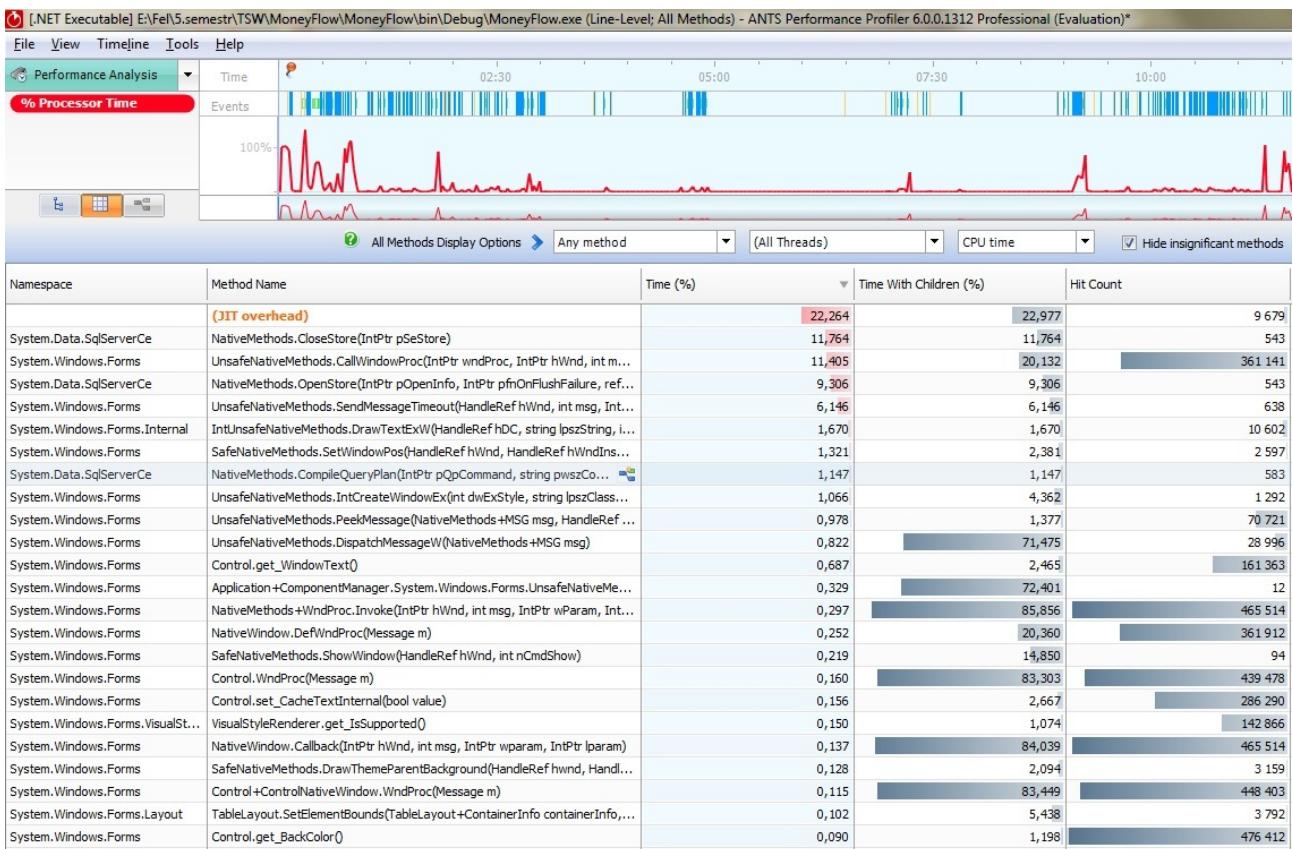
obsahoval vždy duplicitní data. Po opravě tohoto nedostatku a následných testech paměti už bylo vše v pořádku a **problém byl vyřešen**.



Obr. 12 - Zachycení přírustku nových instancí mezi dvěma snapshoty

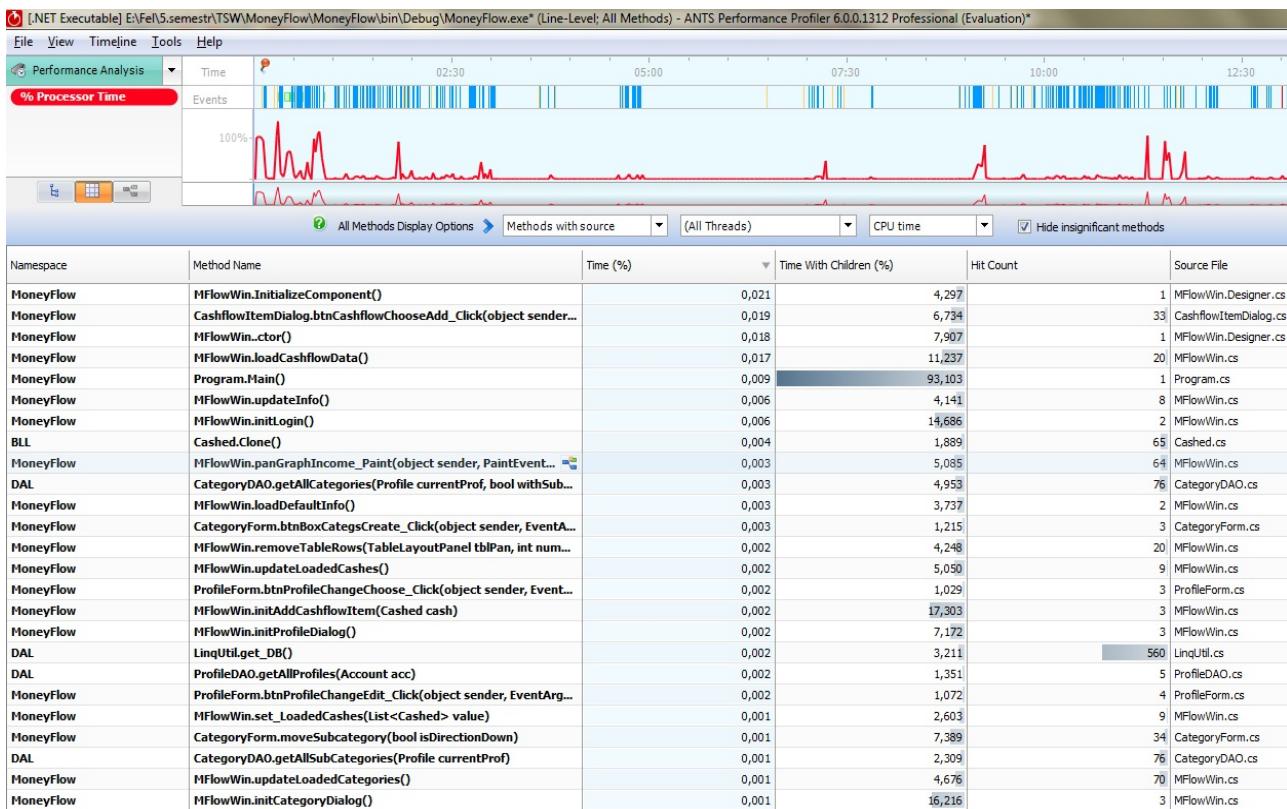
## 5.1 Performance profiling

Běh aplikace zbytečně nezaměstnává procesor – při načítání byl procesor využit asi na 20%, při přidávání položek do databáze potom maximálně na 14 %. Zajímalo nás, jak zaměstnávají procesor jednotlivé funkce volané v programu. K analýze tohoto problému jsme použili program Ants Performer Profiler. Na obrázku 13 vidíme, že „nejžravější“ co se týče času byl C# Garbage collector, dále potom metody z třídy System.

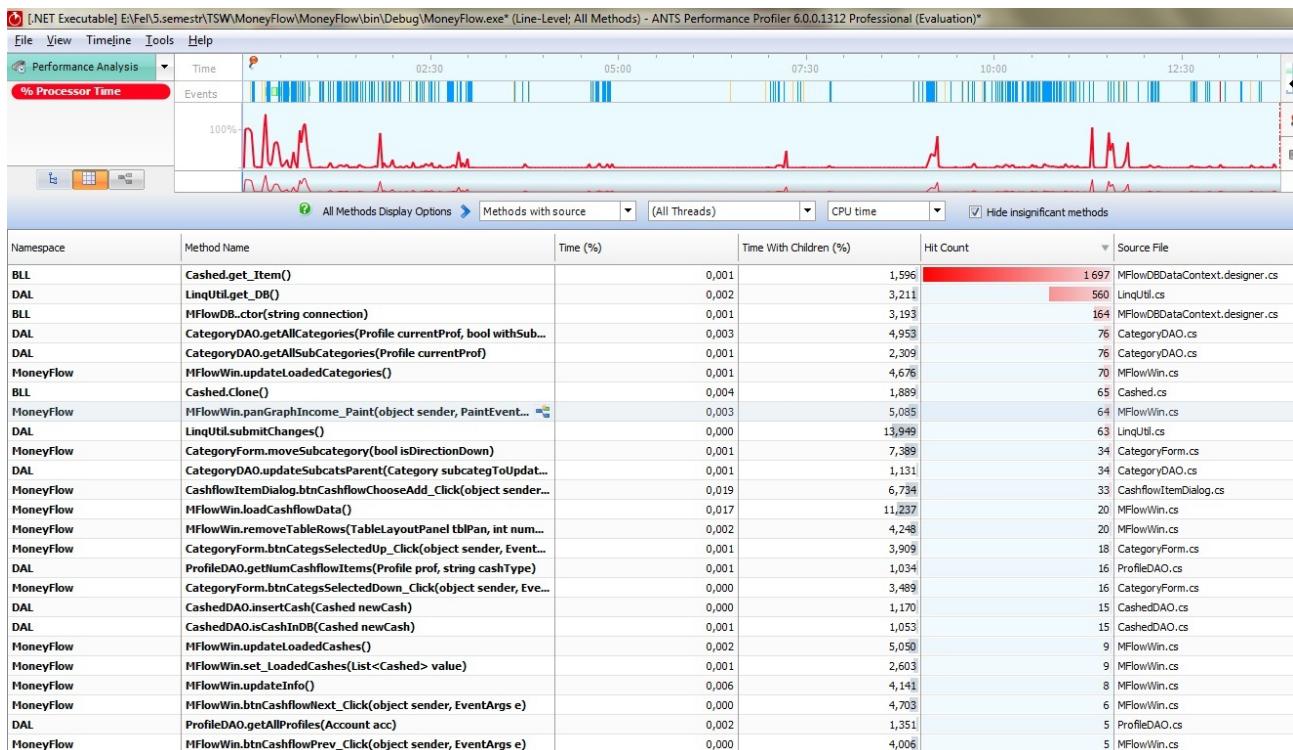


Obr 13. - Performance profiling – všechny metody

K zhodnocení práce tvůrce programu jsme si nechali zobrazit jen metody se zdrojovým kódem – viz obrázek 14. Z hlediska vývoje programování pro nás byl výsledek zajímavý – program trávil v části se zdrojovým kódem aplikace celkově asi 1% času. Nicméně jediné, co nám tento test řekl bylo, že se vývojář spoléhá na již hotové knihovny. Proto jsme začali sledovat, kolikrát byla která metoda volána jako zobrazuje obrázek 15, sloupec hit count. K případnému zvýšení efektivity je potřeba profilovat části programu, odkud se volají metody Cashed.get\_item() a LinqUtil.get\_DB().



Obr. 14 - Performance profiling částí programu se zdrojovým kódem



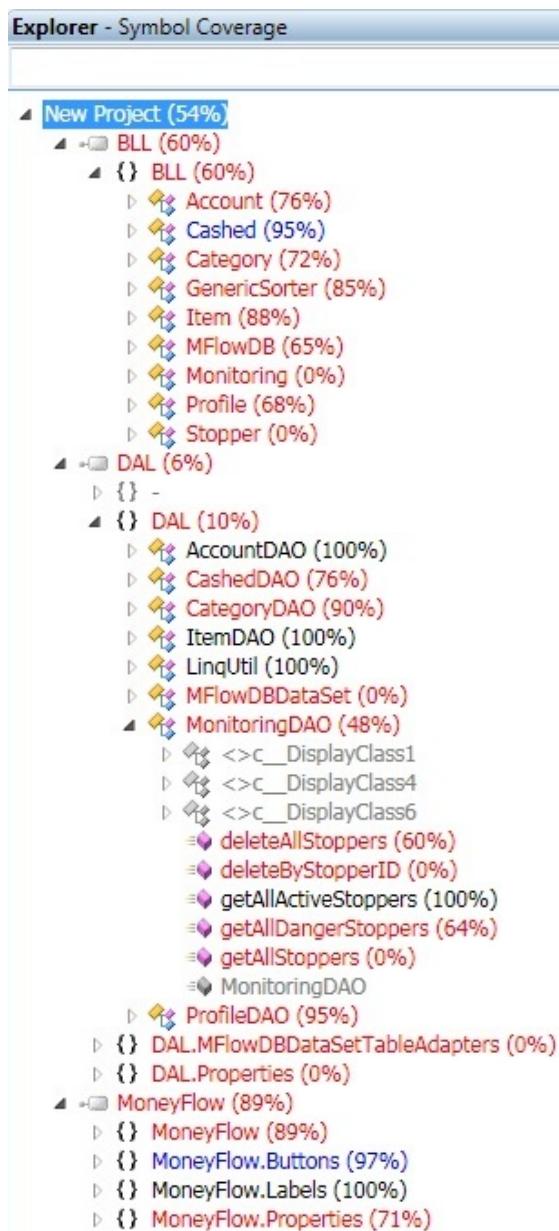
Obr. 15 - Hit count (počet volání) jednotlivých metod

## 6.1 Code Coverage (pokrytí kódu)

K nalezení takzvaných mrtvých částí kódu (nepoužívaný kód – místa, kam se tok řízení programu nedostane) jsme použili program NCover.

Aplikaci jsme spustili přes NCover a otestovali veškerou jeho funkcionality. Code coverage se dá rozdělit podle řady kritérií na Function/Method Coverage, Statement/Symbol Coverage, Decision Coverage/Branch Coverage a Condition Coverage.

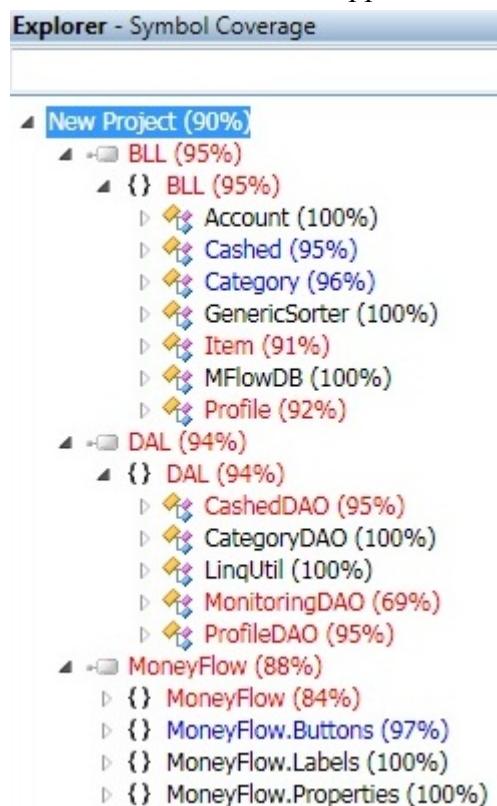
Konkrétně jsme se zaměřili na Symbol Coverage, Branch Coverage a Method Coverage. Symbol coverage, jinak také Statement Coverage je vhodný k nalezení bloků kódu, které nebyly provedeny. Pokud si představíme tok řízení programu jako orientovaný graf, tak nám jednotlivé statementy představují Nody – uzly, přes které program prošel. Branch (Decision) Coverage se zaměřuje na pokrytí hran takového grafu a je vhodný k testování/kontrole pokrytí větví if, if-else a switch. Method Coverage zobrazuje procentuální průchod metodami programu.



Obr. 16 - Symbol/Statement Coverage

Výsledek byl překvapující. Statement Coverage (*obrázek 16*), Decision Coverage i Method Coverage nám zobrazily nevelké pokrytí některých modulů. Program je rozdělen do tří modulů BLL (business layer), DAL (data layer) a Moneyflow (prezentation layer) – jádro a řízení programu.

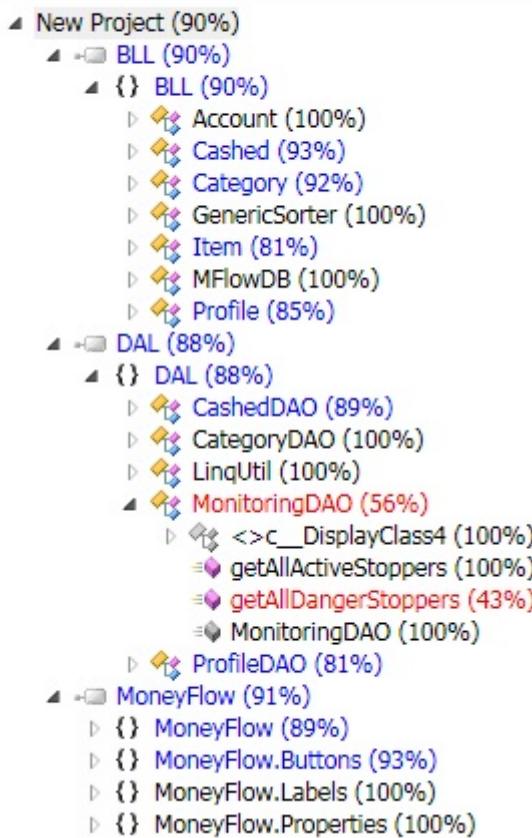
Program odhalil několik tříd, které vůbec nebyly využity. Zjistili jsme, že to jsou třídy spojené s nyní ještě neimplementovanou funkcionalitou Stopperů (kvůli low priority ve funkčních požadavcích), dále pak třídy, které se v programu objevili kvůli pomocné logice třídy LinqUtil (MflowDBDataSetTableAdapters atd.), kterou využívá pouze Visual Studio. Proto jsme v programu nechali skrýt třídy, které program vůbec nenavštíví – výsledek byl uspokojující (viz *obrázek 17*). V třídě MoneyFlow program neprovedl příkazy v try-catch blocích, v DAL snižuje procento pokrytí třída MonitoringDAO, která se stará o ukládání stopperů do databáze a o jejich „monitorování“.



Obr. 17 - Symbol coverage s skrytými nenavštívenými třídami

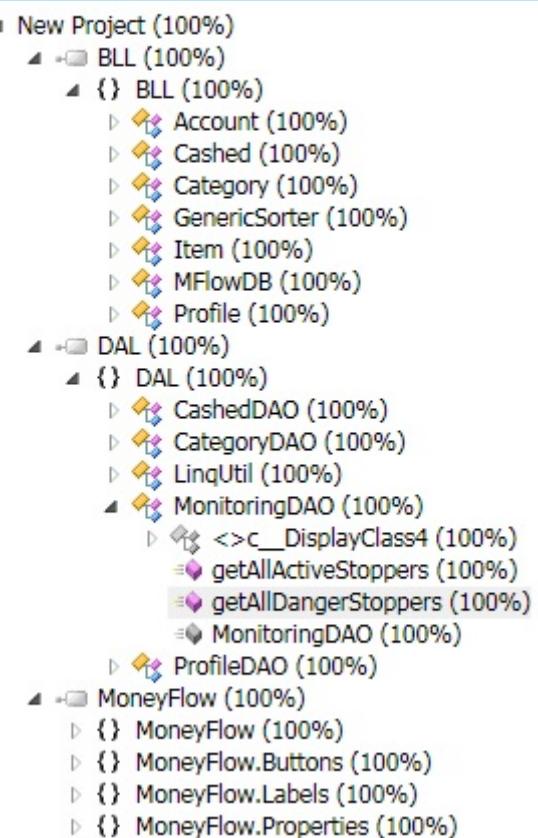
S přihlédnutím k závěrům ze Statement Coverage jsme si zobrazily výsledky Method Coverage a Branching Coverage s skrytými třídami, které by v budoucnu měly zajišťovat lepší funkcionalitu. *Obrázky 18 a 19* ukazují, že metody v modulech i obsažené příkazy if, if-else a switch jsou **pokryté okolo 90 %**, což je docela dobré.

### Explorer - Branch Coverage



Obr. 18 - Branching Coverage

### Explorer - Method Coverage



Obr. 19 - Method Coverage

## 8.1 Manual testing

Pomocí Testing manageru, integrované součásti Visual Studio 2010, jsme otestovali TC01 Vytvoření nového uživatele, TC02 Přihlášení do systému a TC03 Editace položek. Na obr. 20 jsou vidět kroky TC01 – krok jedna, který je mimo obrazovku zahrnuje spuštění programu s očekávaným výsledkem zobrazení účtů, které jsou již v databázi.

The screenshot shows the Microsoft Test Manager application window. The title bar reads "Testing Center". The "Test" tab is selected. Below the tabs, there are links for "Run Tests", "Verify Bugs", and "Analyze Test Runs". On the right side, there are buttons for "Save and Close", "New", and "Open Items (6)". The main area displays a "Test Result for Vytvoření nového uživatele - 19.11.2010 16:41:33". The result table lists 16 steps, each with an expected outcome:

Krok	Očekávaný výsledek
2	Expected - Zobrazení formuláře pro vyplnění přihlašovacích údajů
3	Expected - Zobrazení chybové hlášky a žádosti o zadání hesla
5	Expected - Zobrazení chybové hlášky a smazání textu v textboxech Password a Repeat Password
7	Expected - Zobrazení chybové hlášky oznamující, že zadaná hesla nejsou shodná
9	Expected - Zobrazení chybové hlášky oznamující, že zadaná hesla nejsou shodná
11	Expected - Zobrazení okna s seznamem uživatelských účtů, v němž se nachází i vytvořený účet
13	Expected - Zobrazení formuláře pro vytvoření uživatelského účtu
14	Expected - Zobrazení chybové hlášky upozorňující, že user s tímto nickem již existuje

Obr. 20 – Manualní testování

Z obrázku je viditelné, že program testem úspěšně prošel a správně reagoval na všechny validní i nevalidní vstupy :

1. Uživatel s novým nickem, heslo s 6 a více znaky
2. Duplicítní uživatel
3. Chyba při opisování hesla
4. Krátké heslo

The screenshot shows the Testing Center interface with the following details:

- Top Bar:** Includes icons for back, forward, home, and search, followed by "Testing Center" with a dropdown arrow, "Plan", "Test" (which is highlighted in yellow), "Track", and "Organize". On the far right, it says "MoneyFlow TC".
- Left Sidebar:** Buttons for "Run Tests", "Verify Bugs", and "Analyze Test Runs".
- Right Sidebar:** Buttons for "New" (with a dropdown arrow) and "Open Items (3)".
- Main Content Area:**
  - Section Header:** "Run Tests" with a green play button icon.
  - Build Information:** "Build: Build unavailable" with a dropdown arrow.
  - Test Suite Selection:** "Test suite: MoneyFlow TC" with a dropdown arrow.
  - Tool Buttons:** "Run" (dropdown arrow), "View results", "Open test case", and others.
  - Table View:** A grid showing test cases. The columns are: Order, ID, Title, Tester, Configuration, Priority, and Automated. The first row is a header, and the second row is a summary for "Passed (3)".

Order	ID	Title	Tester	Configuration	Priority	Automated
Passed (3)						
1	1	TC01 Vytvoření nového uživatele	woxie	Windows 7 and IE 8	3	No
2	3	TC02 Přihlásit do systému	woxie	Windows 7 and IE 8	2	No
3	4	TC19 Editace Položek	woxie	Windows 7 and IE 8	3	No

Obr. 21 Výsledek manuálního testingu

Obr. 21 zobrazuje výsledek našeho testingu. TC02 (obr. 22) testuje, zdali systém nechává zalogovat uživatele, kteří správně zadají heslo a zda při špatném zadání hesla zobrazuje chybovou hlášku.

Testing Center | Plan | Test | Track | Organize

Run Tests | Verify Bugs | Analyze Test Runs

Save and Close | New | Open Items (4)

## Test Result for TC02 Přihlásit do systému - 19.11.2010 18:49:26

Summary (Passed)

Details

Analysis

Test Step Details

- 1 Vyber účet, k němuž znáš heslo, případně vytvoř nového usera (viz UC01 vytvoření nového usera) (i)
- 2 Klikni na vybraný účet (checkmark)  
Expected - Textbox Password se stane editovatelným
- 3 Zadej heslo, které nepatří k danému účtu (i)
- 4 Klikni na tlačítko Log in (checkmark)  
Expected - Zobrazení chybové hlášky informující o nesprávně zadaném heslu
- 5 Zadej heslo, které patří k danému účtu (i)
- 6 Klikni na tlačítko Log in (checkmark)  
Expected - Přihlášení do systému

Attachments (3)

Links (0)

Result History (1 Results)

Create bug | View

Result	Created date	Failure type	Resolution	Notes	Run by
Passed	19.11.2010 18:49:26	None	None		woxie

woxie

Obr. 22 Zalogování do systému

Poslední test case – TC19 (*Obr. 23*) zobrazuje kontrolu správné funkčnosti editace položky.  
Zachycené situace:

1. Změna typu položky
2. Změna příslušnosti položky k Category
3. Nevalidní – „prázdná“ Category
4. Změna hodnoty položky na číselnou hodnotu
5. Nevalidní změna hodnoty položky na nečíselnou hodnotu
6. Nevalidní změna hodnoty položky na „prázdnou“ hodnotu (řetězec „“)

The screenshot shows the Testing Center application interface. The top navigation bar includes icons for back, forward, home, and a dropdown menu labeled "Testing Center". Below the menu are buttons for "Plan", "Test" (which is highlighted in yellow), "Track", and "Organize". On the far right, it says "MoneyFlow TC" and shows "New" and "Open Items (5)".

The main area displays the "Test Result for TC19 Editace Položek - 19.11.2010 18:49:26". The result is marked with a green checkmark. The test steps are listed on the left, and their expected outcomes are shown in blue boxes on the right.

Test Step	Expected Outcome
7 Změň hodnotu Value na 1a23	Expected - Zobrazení chybové hlášky upozorňující na špatný formát hodnoty value, text v poli Value změněn na původní hodnotu
8 Klikni na tlačítko Save changes	Expected - Zobrazení chybové hlášky upozorňující na špatný formát hodnoty value, text v poli Value změněn na původní hodnotu
9 Změň popisek	
10 Klikni na tlačítko save changes	Expected - Zavření okna s detaily položky, změna popisku položky v seznamu na zadanou hodnotu
11 Klikni na menubar Myflow	Expected - zobrazení podseznamu
12 Vyber položku Categories	Expected - zobrazení Okna s kategoriemi
13 Zadej název nové kategorie	
14 klikni na create Category	Expected - Zobrazení hlášky o vytvoření kategorie
15 Vyber položku a klikni na ikonu tužky	Expected - Zobrazení okna s detaily položky
16 V checkboxu To Category vyber hodnotu: -Choose Category-	
17 Klikni na tlačítko save changes	Expected - Zobrazení chybové hlášky upozorňující na nevybranou Categorii, v checkboxu To Category je zpět původní hodnota
18 V checkboxu vyber jméno categorie z bodu 13	
19 Klikni na tlačítko save changes	Expected - Zavření okna s detaily položky, změna category položky v seznamu na zadanou hodnotu

*Obr 23.* Editace položek

## 9.1 Unit Testing

Pro jednotkové testování byl použit integrovaný modul Visual Studio 2010 Ultimate určený pro Unit Testy. Jednotkovými testy jsme se rozhodli pokrýt manipulaci s uživatelskými účty, tedy editaci uživatele, vytvoření uživatele a smazání uživatele, konkrétněji:

- **Funkce: Nový uživatelský účet**
  - Vytvoření nového uživatelského účtu
  - Validaci vkládaných dat a správné vyhození výjimek při špatně zadaných údajích (jako dlouhé nebo krátké heslo, prázdný řetězec ve jménu atd.)
  - Ověření hesla účtu
  - Konverzi objektu účtu do řetězcové reprezentace
  - Chování při pokusu vytvořit účet, který již existuje
- **Funkce: Změnit údaje uživatelského účtu**
  - Změny údajů v již vytvořeném uživatelském účtu (update)
  - Nemožnost změnit ID účtu pokud je uložen v databázi.
  - Chování při pokusu změnit údaje účtu na údaje, které již v systému existují
- **Funkce: Smazat uživatelský účet**
  - Smazání uživatelského účtu
  - Chování při pokusu smazat účet, který v databázi neexistuje

Softwarová architektura aplikace je postavená na 3 vrstvách – datové, byznys a prezentační. Zde vznikl problém jak efektivně otestovat funkcionality popsanou funkčními požadavky, protože prezentační vrstva je vůči okolí zapouzdřená a téměř (s párem výjimkami) neposkytuje žádné veřejné metody. Prezentační vrstva skládá dohromady instance z datové vrstvy, která slouží pro ukládání objektů z byznys vrstvy na datové uložiště (konkrétně do databáze), a z byznys vrstvy, která realizuje model Moneyflow aplikace a funkcionality, takže nám stačí testy pokrýt právě tyto dvě vrstvy a máme otestováno (*Obrázek 24*).

```
ng.NewUserTest
{
    // Use TestCleanup to run code after each test has run
    // [TestCleanup()]
    // public void MyTestCleanup() { }
    //
    #endregion

    [TestMethod]
    public void TestCreateNewUser()
    {
        // Adding new User to Database on DAL layer
        testAccDAO.insertAccount(testAcc);

        // Committing changes (= Save into real DB)
        LinqUtil.submitChanges();
        // Clean the mess
        LinqUtil.cleanUp();

        // STARTING TEST!
        // Try to get model of same ID from DB
        Account outputAcc = new AccountDAOTesting().getAccountByID(testAcc.Account_id);

        Assert.AreEqual(testAcc.Account_id, outputAcc.Account_id);
        Assert.AreEqual(testAcc.Name, outputAcc.Name);
        Assert.AreEqual(testAcc.Nick, outputAcc.Nick);
        Assert.AreEqual(testAcc.Pass, outputAcc.Pass);
    }

    [TestMethod]
    public void TestCreateNewUserByNameValidationFail()
    {
        // Create new User with invalid name
        string invalidName = "invalid_name";
        Account testAcc = new Account();
        testAcc.Name = invalidName;
        testAcc.Nick = "nick";
        testAcc.Pass = "password";
        testAcc.Account_id = null;

        try
        {
            testAccDAO.insertAccount(testAcc);
        }
        catch (Exception ex)
        {
            // Check if validation error was thrown
            Assert.AreEqual("Validation error", ex.Message);
        }
    }
}
```

*Obr. 24 – Ukázka kódu unit testů*

Rovněž jsme se rozhodli nepoužívat mock a stab objekty, protože bychom náhradou některé z reálných vrstev mohli přijít o testy jejich vzájemného propojení, které je však klíčové pro správnou funkcionality – testujeme tedy i správné „probublání“ dat na datovou vrstvu a zpět. Naše unit testy testují jak správnou funkci, tak správné vyhození výjimek v určitých situacích. Dále se testy snažíme pokrýt co nejvíce procent kódu testovaných tříd (code coverage, obrázek 25).

Hierarchy	Not Covered (Bl...)	Not Covered (% Blocks)	Covered (Blocks)	Covered (% Blocks)
NG@NG-PC 2010-11-11 19:16:09	2387	85,34%	410	14,66%
DAL.dll	1370	93,07%	102	6,93%
{} DAL	1370	93,07%	102	6,93%
▷ CashedDAO	380	100,00%	0	0,00%
▷ CategoryDAO	212	100,00%	0	0,00%
▷ ItemDAO	11	100,00%	0	0,00%
▷ MonitoringDAO	172	100,00%	0	0,00%
▷ ProfileDAO	578	100,00%	0	0,00%
▷ AccountDAO	17	41,46%	24	58,54%
▷ AccountDAOTesting	0	0,00%	60	100,00%
▷ LinqUtil	0	0,00%	18	100,00%
BLL.dll	989	90,40%	105	9,60%
{} BLL	989	90,40%	105	9,60%
▷ AccountTesting	11	100,00%	0	0,00%
▷ Cashed	120	100,00%	0	0,00%
▷ Category	303	100,00%	0	0,00%
▷ GenericSorter	20	100,00%	0	0,00%
▷ Item	78	100,00%	0	0,00%
▷ Monitoring	196	100,00%	0	0,00%
▷ Profile	147	100,00%	0	0,00%
▷ Stopper	76	100,00%	0	0,00%
▷ MFlowDB	24	77,42%	7	22,58%
▷ Account	14	12,50%	98	87,50%

Obr. 25 – Pokrytí kódu testovaných tříd (code coverage)

Při spuštění testů v současnosti všechny procházejí v pořádku a bez chyb (viz Obrázek 26).

NG@NG-PC 2010-11-11 19:16:09			
Test run completed Results: 13/13 passed; Item(s) checked: 0			
Result	Test Name	Project	Error Mess
Passed	TestAccountToString	UnitTesting	
Passed	TestCreateNewUser	UnitTesting	
Passed	TestCreateNewUserNameValidationFail	UnitTesting	
Passed	TestCreateNewUserNickValidationFail	UnitTesting	
Passed	TestCreateNewUserPassValidationFail	UnitTesting	
Passed	TestCreateSameUserFail	UnitTesting	
Passed	TestDeleteExistingUser	UnitTesting	
Passed	TestDeleteNotExistingUserFail	UnitTesting	
Passed	TestEditUserInDB	UnitTesting	
Passed	TestEditUserInDBSameNickFail	UnitTesting	
Passed	TestEditUsersIDFail	UnitTesting	
Passed	TestCheckPasswords	UnitTesting	
Passed	TestCheckPasswordsFail	UnitTesting	

Obr. 26 – Výsledky unit testů