Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra softwarového Inženýrství

Bakalářská práce

Vývoj software s využitím moderních vývojových prostředků

Petr Prchal

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci Vývoj software s využitím moderních vývojových prostředků jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne datum odevzdání _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Jiřímu Brožkovi, Ph.D., za pozornost a pomoc, kterou věnoval mé práci, a za jeho odborné rady, které pomohly k vypracování této práce.

Vývoj software s využitím moderních vývojových prostředků

Software development using modern development tools

Souhrn

Tato práce se zabývá problematikou vývoje softwaru, a to především s přihlédnutím k možnostem využití platformy .NET Framework. Obsah celé práce jde v podstatě rozdělit do 3 hlavních částí.

První část bude zaměřena primárně na stručný přehled vývoje programování od sekvenčního programování až po objektové. V této části budou vyzdvižené některé společné vlastnosti jednotlivých programů a určité důležité prvky, které přetrvávají i v dnešních programovacích jazycích. Právě v této části budou nastíněny některé důležité vlastnosti objektového programování.

Druhá část bude zaměřena na jazyk C#, problematiku .NET Framework, ale i implementaci Mono. Také právě v této kapitole budou rozebrány dvě vývojové prostředí a to Microsoft Visual Studio a Xamarin studio.

Ve finální části bude zkonstruována vzorová aplikace za použití obou prostředí, přičemž aplikace v prostředí Visual Studio bude zaměřena na platformu Windows a aplikace v prostředí Xamarin Studio bude zaměřena na mobilní platformu Android. V této části budou vyzdviženy rozdíly tvorby stejné aplikace, za pomoci programovacího jazyka C# pomocí zmíněných rozdílných prostředí a platforem.

Klíčová slova: aplikace, framework, .Net, programovací jazyk, software, objektové programování, integrované vývojové prostředí, třída, metoda

Summary

This bachelor thesis is engaged in software development issues especially with regard to the possibility of using .NET Framework. The content of the thesis is in fact divided into three main parts.

The first part will focus primarily on a brief overview of programming from sequential programming, to object. This section will be highlighted some of the common features of various programs and some important elements that persist in today's programming languages. It is in this section will outline some of the important features of object-oriented programming.

The second part will focus on the C # language, the issue NET Framework, but also the implementation Mono. Also, just in this chapter will be discussed in two development environments and Microsoft Visual Studio and Xamarin studio.

In the final section will be developed sample application using both programming environments, with application in Visual Studio will be focused on the Windows platform and application in Xamarin Studio will be focused on the mobile platform Android. This section will be highlighting differences in the formation of the same application, with the help of the programming language C #, using different environments and platforms.

Keywords: aplication, framework, .Net, programing language, software, object programing, Integrated development enviroment, class, method

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Pod pojmem vývoj software se neskrývá pouze programování, ale celý životní cyklus vývoje nové aplikace, který zahrnuje také dokumentaci, užívání nástrojů nutných pro vývoj programu a testování námi vytvořeného programu. Vývoj software a programování byly vždy nedílnou součástí informačních technologií a pro tvorbu software bylo vytvořeno mnoho programovacích jazyků, mezi které patří například Pascal, ALGOL, Java, ale také jazyk C a C++.

Právě jazyk C byl jedním z programovacích jazyků, které nejvíce zasáhly do rozvoje programování. V této práci se budeme zabývat také problematikou .NET a prakticky výhradně se zaměříme na programovací jazyk C#, který je vlastně potomek jazyka C a C++.

Nejznámějším a nejrozšířenějším editorem jazyka C# je v dnešní době beze sporu Visual Studio od společnosti Microsoft. Tento editor je však primárně zaměřen na tvorbu pro platformu Windows. V dnešní době je však vysoký nárůst poptávky po mobilních aplikacích, které jsou vytvořeny pro smartphone, iPad, iPhone či tablet. Proto se v této práci podíváme kromě MS Visual Studia také na editor od společnosti Xamarin, který je zaměřený na multi-platformní tvorbu aplikací s primárním cílem pro mobilní aplikace, za použití programovacího jazyka C#.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem této práce je analyzovat problematiku vývoje software s přihlédnutím na využití technologie .NET Framework a programovacího jazyka C#. V rámci této práce bude zmapován vývoj programování a problémy, které byly spojeny s jeho vývojem a seznámení se s moderními vývojovými prostředími, pracujícími s platformou .NET.

Na základě zjištěných poznatků bude vytvořena vzorová aplikace za pomoci programových prostředků, dále bude nutností se seznámit s možným řešením a ukázat rozdíly tvorby aplikace mezi jednotlivými vývojovými prostředími za použití jazyka C#.

2.2 Metodika práce

Metodika bakalářské práce je založena na studiu a analýze odborných informačních zdrojů. Jednotlivé zdroje byly opatřeny z tištěné literatury a různých internetových zdrojů. Z důvodu rychlého vývoje v tomto oboru je většina zdrojů čerpána z internetových zdrojů, a to převážně z knihoven společnosti Microsoft. Za pomoci uvedených zdrojů byla tato práce napsána a sestavena.

Praktická část práce spočívá ve využití programových prostředků se zaměřením na platformu .NET a v menší míře i platformu Mono.

Na základě syntézy teoretických poznatků a výsledků praktické části pak byly formulovány závěry bakalářské práce.

3 Vývoj programování a základní pojmy objektového programování

3.1 Začátky a vývoj programování

Předtím než se budeme zabývat přímo jazykem C# a platformou .Net, je podle mého názoru dobré udělat si stručné popsání vývoje programovacích jazyků, abychom věděli, jak jsme se dostali od starých jazyků sekvenčních programovacích jazyků, jejichž typickým zástupcem byl například assembler až po dnešní objektové programovací jazyky, mezi které například patří C#, ale i další jazyky jako například Java, PHP a podobně.

Za základy počítačového programování se dá považovat přepojování jednotlivých drátů na systému Eniac. První programovací jazyk existoval jako výhradně strojový kód, který vyjadřoval strojové instrukce, proto se často nazývá strojový jazyk. Sémantika instrukcí strojového jazyka byla definována pomocí jednotlivých technických součástí počítače, z čehož vyplývala závislost na konkrétním typu počítače. Jedním z dalších hlavních problémů byla náročnost uživatele na zápis jednotlivých číselných instrukcí do počítače. V dnešní době se tento způsob programování již prakticky nevyužívá. (Čajda, 2009) (Kučera, 2000)

Důležité je také rozlišit programovací jazyky na procedurální a neprocedurální. V dnešní době je naprostá většina jazyků procedurální. Procedurální jazyky mají jeden společný rys, a to jednoznačné popsání řešení dané úlohy. Jedním ze stavebních kamenů těchto jazyků je označení místa v paměti jako proměnné a také označení přiřazovacího příkazu, který nám zpřístupní námi získanou informaci, hodnotu, apod. a dále umožňuje ji uložit do proměnné. Jak již bylo řečeno, v dnešní době je naprostá většina programovacích jazyků procedurální, a to například Java, Pascal, C, C++, C#, ale i assembler, basic a podobně. (Čajda, 2009)

V neprocedurálních (deklarativních) jazycích sice specifikujeme úlohu, kterou se chystáme vyřešit, nikoliv však přesný způsob řešení této úlohy. Neprocedurální jazyky vlastně popisují pouze cíl, kterého chceme dosáhnout. Mezi neprocedurální jazyky se řadí funkcionální (například Haskell) jazyky a logické jazyky (prolog). Neprocedurálním jazykům se však věnovat nebudeme. (Čajda, 2009) (Merunka, 2006)

3.2 Sekvenční programování

Jedním z prvních pokroků ve vývoji programovacích jazyků bylo vyměnění numerického zadávání informací ve strojovém jazyce za symbolické zadávání informací. Sekvenční programování probíhalo tak, že v určitém kódu programovacího jazyka vývojář psal sekvenci příkazů, které počítač prováděl a postupně zpracovával. Samozřejmě byly možnosti, pomocí kterých bylo možné narušit sekvenci a řízení přesunout jinam. Problémem bylo, že tyto možnosti byly velmi nepohodlné a s přihlédnutím na pozdější úpravy jednotlivých programů také nešikovné. Prakticky všechny byly založeny na primitivních podmínkách a přesunech na odlišná místa v určité sekvenci příkazů. (Merunka, 2006) (Čajda, 2009)

Jak již bylo zmíněno, nejznámějším zástupcem sekvenčních programovacích jazyků je Assembler (překladač). Vznik Assembleru se datuje někdy do konce čtyřicátých let minulého století. Jedná se o jazyk symbolický. V současné době se tento nízkoúrovňový programovací jazyk používá téměř výhradně pro přímé ovládání hardware, a to z důvodu že assembler umožňuje vysoký výkon, například u ovládání instrukcí procesoru nebo u určitých ovladačů zařízení.

Výhody assembleru:

- programy se dají napsat poměrně rychle a nejsou náročné na výkon
- umožňuje vytvoření téměř libovolného programu

Nevýhody assembleru

- Je poměrně složitý a uživatelsky náročný (složitost spočívá ve vysokém počtu jednotlivých příkazů)
- V dnešní době je již zastaralý
- Zdrojové kódy, které jsou napsány v tomto jazyku, jsou často velmi nepřehledné

3.3 Strukturované programování

Strukturované programování se začalo používat někdy koncem 70 let minulého století a hlavním motivem jeho vývoje byl důvod, že vývojáři software dospěli k názoru, že předávání jedné rozsáhlé dávky příkazů nebo určité sítě prvků vývojového diagramu je poměrně komplikované a nedostatečně efektivní. Další podstatný důvod vývoje strukturálních programovacích jazyků byla také vysoká složitost úprav nebo opravy chyb v kódu

sekvenčních programovacích jazyků. Za tvůrce tohoto konceptu programování je považován Edsger W. Dijkstra. Vývojáři se proto rozhodli vytvořit techniku, kde rozdělí úlohu na dílčí procedury tvořící jeden celek. Dá se také říci, že strukturované programování se zakládá na těchto principech: (Kučera, 2000)

- Náročnou úlohu rozdělíme na menší dílčí bloky, Tyto bloky poté zpracováváme jednotlivě a v závěru je propojíme v jednu komplexní úlohu zpravidla metodou shora dolů.
- Pokud konstruujeme řešení jednotlivých dílčích bloků, musíme používat výhradně povolené řídící struktury.

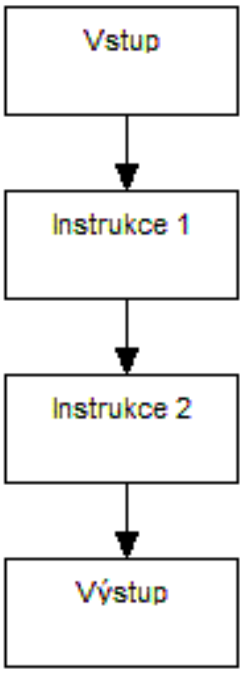
Z těchto principů vyplývá, že na místo předávání řízení sekvence jinam ve stylu sekvenčních jazyků je program rozdělen na jednotlivé procedury, které se například ptají "nastane-li podmínka 1, proveď tuto proceduru, pokud ne, proveď vnořenou proceduru 2", nebo "Proveď podmínku 1 (a po každém provedení přičti 1 do cyklu) tolikrát, dokud se nedojde k námi určenému číslu". Sekvence příkazů, která nám vzniká v procedurálních programovacích jazycích, je sice na první pohled téměř stejná jako v jazycích sekvenčních, ale neobsahuje přeskoky a podstatně méně pomocných podmínek. (Čajda, 2009) (Kučera, 2000)

Z těchto informací vyplývá, že strukturovaně navrhovaný algoritmus je výrazně méně chaotický a výrazně přehlednější než algoritmus sekvenční. Přestože obsahuje pomocné podmínky, tento algoritmus neobsahuje skoky z jedné části programu do jiné (například skok ze začátku na konec a zpět). Z důvodu, že všechny bloky strukturovaně navrženého programu mají jasně definované vstupy a výstupy, má za následek, jak již bylo řečeno, výrazné zjednodušení možnosti oprav chyb a úprav programu oproti sekvenčnímu způsobu. Pro samotné úpravy totiž není ve většině případů upravovat celý kód programu, ale stačí pouze upravit jednu část (blok), ve kterém je chyba. (Kučera, 2000)

Samotný životní cyklus programu zpravidla začíná inicializací projektu a postupně přejde k načítání dat. Jakmile jsou data načtena, program začne s jejich zpracováváním a výstupu výsledků projektu. Každé prostředí, ve kterém se programy spouštějí, se samozřejmě liší, ale základní struktura a některé jejich vlastnosti zůstávají stejné, například: zavaděč programu do operační paměti, definování dat, se kterými se chystáme pracovat a samotné příkazy. (Gauld) (Kučera, 2000) (Čajda, 2009)

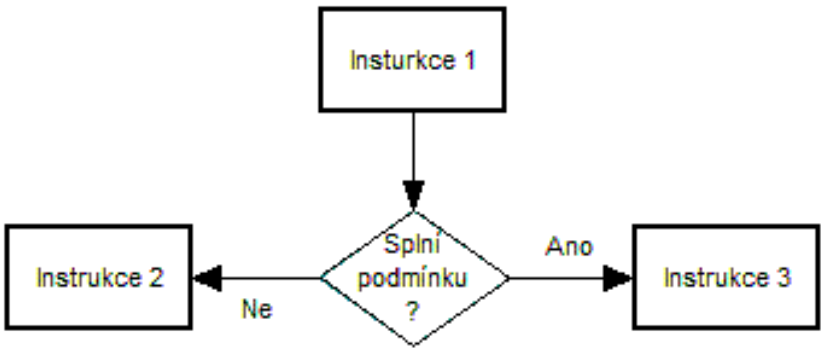
Nyní je na čase podívat se na hlavní společné vlastnosti strukturálních programů:

- Programy musejí být schopny provádět jednotlivé instrukce v přesném námi určeném pořadí.



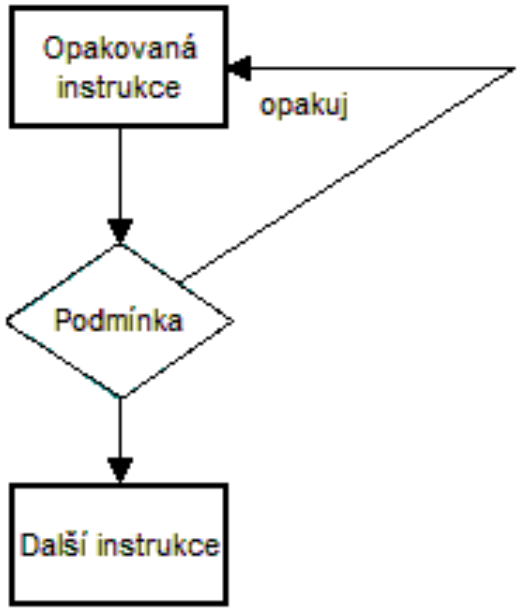
Obrázek 1: Schéma posloupnosti instrukcí (vytvořeno vlastním zpracováním na základě (Gauld)*)*

- Pokud v průběhu instrukce dojde k nějaké podmínce rozhodování, program musí být schopen rozlišit, zda-li je podmínka splněna, a podle toho zareagovat. Tomuto větvení se často říká podmíněné větvení. (Gauld)



Obrázek 2: Schéma Větvení (vytvořeno vlastním zpracováním na základě (Gauld)*)*

- Pokud se jedná o cyklus, program musí být schopen provádět stejnou operaci do doby, dokud se nenaplní podmínka určující konec cyklu. Jakmile tato podmínka nastane, program se posune na další instrukci, popřípadě se ukončí. (Gauld)



Obrázek 3: Schéma Cyklu (vytvořeno vlastním zpracováním na základě (Gauld)*)*

- Pokud program provádí určitou posloupnost úloh několikrát za sebou, máme možnost umístit tyto úlohy do funkce (často nazýváno také modulu). Tato funkce se poté chová jako podprogram, který je spustitelný z námi navrhovaného programu. (Gauld)

Strukturální programování bylo vysoce úspěšné a životaschopné. Programovací jazyky využívající tento styl byly u programátorů populární po poměrně velkou dobu, i v dnešní době je tento styl programování poměrně populární a téměř absolutní většina programovacích jazyků má kořeny a základní principy právě ze strukturovaného programování a jeho hierarchie. Mezi nejznámější mohu uvést Pascal, ALGOL. Nejspíše nejpopulárnější zástupce je jazyk C, u kterého se dá říci, že se z něj postupně vyvinul jazyk C#, kterému se budeme v této práci věnovat.

Postupem času programátoři dospěli k názoru, že ač je přesná hierarchická struktura programu užitečná ve většině případů, příliš nám nepomáhá, pokud vytváříme úlohu, kde potřebujeme mít skupinu vzájemně spolupracujících a komunikujících částí, které se podle momentální nutnosti staví do potřebných hierarchií nebo je opouštějí (snaha přiblížit se reálnému světu). Toto byl také jeden z důvodů vzniku objektového programování.

3.4 Objektové programování

Jak strukturované programování nahradilo přesnou sekvenci příkazů výrazně přehlednější a přístupnější hierarchií, objektové programování překonává i hierarchii strukturovaného programování. Objektově orientovaný přístup se poprvé objevil v 60 letech 20 století a prvním programovacím jazykem využívající tento přístup byla Simula 67 (dle roku 1967), která byla odvozena z jazyka Algol. V 70 byl vyvinut ve firmě Xerox PARC jazyk Smalltalk. Právě Xerox začal jako první používat termín objektové programování. Od 70. let jsou služby podporující objektové programování v určité míře téměř ve všech programovacích jazycích. (Čajda, 2009) (Merunka, 2006)

3.4.1 Požadavky kladené na objektové programování

Dá se říci, že hlavními účely a filozofií objektově orientovaného programování je postarat se o vysokou možnost opětovného využití programu, zajistit robustní a funkční program, udělat program maximálně přizpůsobivý k modifikacím, dobře přenosný mezi jednotlivými platformami a pokusit se co nejvíce přiblížit filozofii programu reálnému světu.

- Jedním ze základních snah o vysokou možnost opětovného využití kódu programu je snaha využít stejný kód ve více částech systému, popřípadě i využít mezi několika různými programy, a tím snížit náklady vynaložené na jeho vývoj i složitost orientace ve zdrojovém kódu. Kód, který je velmi často používán, je zpravidla velmi dobře otestován a tím pádem jeho přenesení může být vysoce výhodné. Na druhou stranu pokud takto přenesený kód trpí určitými silnými nedostatky, tak je vysoké riziko, že program, který tento kód přenese, bude trpět stejnými nedostatky.

- Na programování byla vždy kladena snaha o maximální korektnost a funkčnost. Program, který je funkční, stabilní a robustní, by měl být schopen reagovat na

informace, které do něj vložíme správnými výstupy, a pokud námi vložené informace jsou v rozporu s programem, tak by měl být schopen zareagovat i na ně (neměl by se okamžitě ukončit bez udání důvodu, nebo zaseknout). Program by samozřejmě měl být schopen zareagovat i na různé události, které nebyly očekávány při jeho tvorbě, - například ztrátu spojení, námi hledaný soubor neexistuje, nedostatečná operační paměť, nedostatečný výkon procesoru, nebo nekorektní ukončení programu.

- Z důvodu vysokého využití některých programů by měla být umožněna co možná nejjednodušší možnost jeho aktualizování a modifikování. Jeho modifikovatelnost by měla být minimálně riziková a pokud to bude možno, i minimálně náročná. To znamená, že pokud je nutné aktualizovat firemní databázi, nebo firemní server, který je neustále využíván, tak by měla aktualizace proběhnout co nejrychleji, aby firma měla co nejmenší ztráty z doby odstavení serveru, popř. databáze.

- Náš software by mělo být možno také přenést na jinou platformu bez vysokých problémů, například mezi Unixem a Windows.

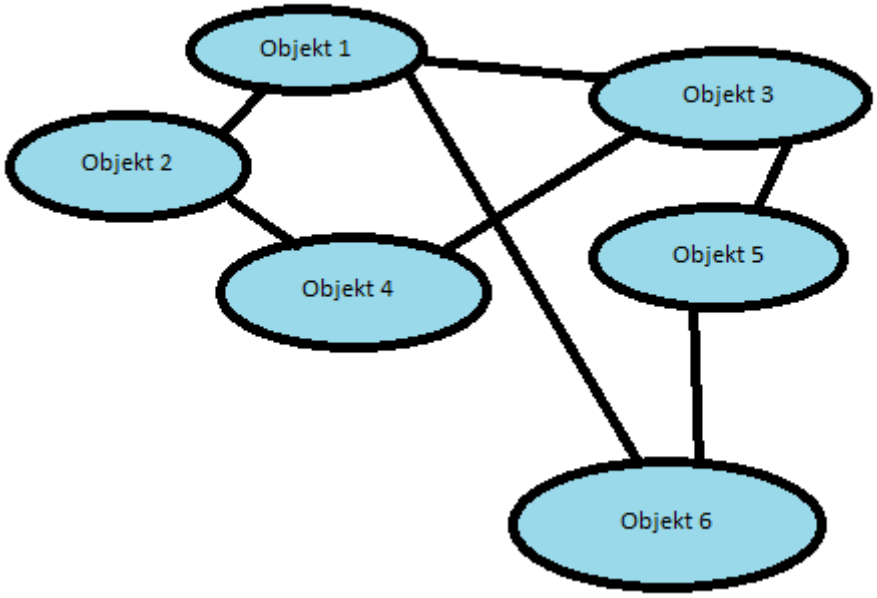
- Objektové programování se snaží maximálně soustředit na zachycení sémantiky reálného světa a klade menší důraz na zpracování dat. Dá se říci, že jednou ze základních koncepcí je uspořádání tříd do určité hierarchie, které se pak přiřazují objekty.

- Software by měl být schopen pracovat na určité úrovni abstrakce. Bohužel zpravidla platí, že čím je systém uživatelsky přístupnější, tím je složitější na naprogramování a naopak.
  (Merunka, 2006) (Marek, 2007)

3.4.2 Důležité prvky objektového programování

Objekt

Základním prvkem programu je objekt, který je ucelený soubor atributů a metod, se kterými také pracuje. Každý objekt musí obsahovat přesně určené rozhraní, které nám dovoluje s ním komunikovat. Jednotlivé objekty mohou představovat nějaký reálný objekt z reálného světa, toto však není podmínkou a objekt může být ryze abstraktní. Objektové programování je tím pádem založeno na síti objektů, které spolu vzájemně komunikují a jsou schopny formovat libovolné struktury. Objekt může obsahovat jiné objekty a může použít vlastnosti jiných objektů, pokud jim dá signál k provedení operace. (Čajda, 2009)



Obrázek 4: Schéma Cyklu (vytvořeno vlastním zpracováním na základě (Čajda, 2009)*)*

Atributy

Atributy (jinými slovy se dají označit jako proměnné programu) jsou jednotlivé data souboru, které soubor uchovává a které objekt identifikují a odlišují od ostatních objektů. Jako příklad se dá uvést například číslo SPZ u auta v seznamu aut. (Merunka, 2006) (Marek, 2007)

Metody

Dá se říci, že metody jsou určité vlastnosti programu, které umožňují programu vykonávat to, co je od něho požadováno. Metody mohou vracet hodnoty a také mohou mít určité parametry.

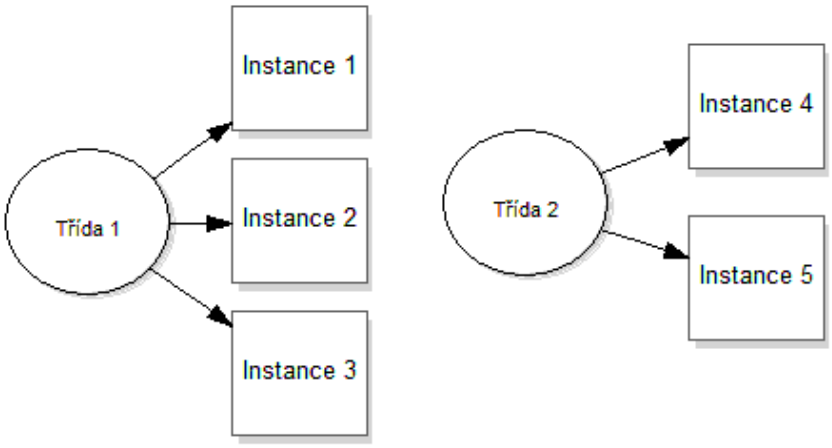
Třída

Pod pojmem třída se dá představit prvek objektového systému, který se dá označit za druh objektu. U všech objektů je jednoznačně určené, ke které třídě patří, a třídy mají definovanou hierarchii, která se snaží zachytit jejich podobné vlastnosti. (Čajda, 2009)

Dá se říci, že třída je určité schéma popisující strukturu objektu a reprezentující druh objektu. Z definice vyplývá, že třídy mají hierarchii, která je definována pomocí dědičnosti, ke které se ještě dostaneme. Některé novější jazyky mají schopnost reflexe. To znamená, že jsou schopny číst struktury tříd, ale i upravovat jednotlivé třídy za běhu programu. Třída nemusí být nutně objekt. Může být určitým unikátním prvkem, který má s objekty společný pouze odkaz. Není to však ve většině případů výhodné, proto je lepší, pokud jsou třídy plnohodnotné objekty. (Čajda, 2009) (Marek, 2007)

Instance třídy

Instancí rozumíme na základě třídy vytvořenou určitou datovou strukturu. Dá se říci, že se jedná o objekt, který není třídou, pokud jednotlivé třídy jsou objekty systému. Instance je však vždy odvozena od určité třídy. Instance se mezi sebou přitom liší svojí adresou paměti a hodnotami svých atributů.



Obrázek 5: Schéma tříd a jejich instancí (vytvořeno vlastním zpracováním na základě (Čajda, 2009)*)*

Abstraktní třída

Abstraktní třída je druh třídy, která slouží jako základ pro další třídy. Přestože v ní není možno vytvářet její instance, dědí z ní třídy, které instance mají. Abstraktní třída nám také umožňuje definovat určité abstraktní metody (nemají implementaci), které však její potomci musejí povinně implementovat. Abstraktní třída se využívá v situaci, kdy pracujeme s objekty skládajícími se z dalších objektů. Jeden objekt se může skládat i z několika tříd objektů. (Merunka, 2006) (Hordějčuk)

Typ

Typ se zpravidla používá pro zlepšení korektnosti námi vytvořeného programu. Jedná se o funkci sloužící pro omezení hodnot, kterých námi určená proměnná může dosahovat. Prvek typ je zpravidla v moderních programovacích jazycích již zahrnut.

3.4.3 Důležité pojmy Objektového programování

Polymorfizmus

Jedná se vlastně o požadavek komunikace jedním stejným způsobem s více objekty najednou (využít jednotné rozhraní pro odlišné typy objektů). Polymorfizmus úzce souvisí se zapouzdřením. Pokud neznáme parametry více zapouzdřených objektů, tak je logické, že budeme se všemi komunikovat stejně.

Polymorfizmus nám dává možnost upravit námi určenou metodu, aby u každé podtřídy reagovala takovým způsobem, jaký si představujeme. Například vytvoříme-li metodu auto v seznamu značek, tak nám podtřídy budou hlásit Škoda, Renault a podobně.

Dědičnost

Dědičnost je způsob tvoření vztahů mezi jednotlivými typy objektů reprezentovaných třídami do určité hierarchie. V této hierarchii je třída, ze které se dědí, označena zpravidla základní třída (nebo také super třída, nadtřída, nadtřída apod.) a nová odvozená třída poté zdědí veškeré vlastnosti této nadtřídy a může je dle nutnosti nadále rozpracovávat.

Pro vytvoření dědění je zpravidla nutné dodržet dvě důležité pravidla. Prvním je, že nově vytvořená třída musí k dispozici metody své nadtřídy a přitom je může dle své nutnosti

upravovat a měnit. Druhé pravidlo říká, že instanční proměnné základní třídy se musejí stát součástí objektů nově vytvořené třídy.

V některých programovacích jazycích je možné vytvoření neděditelné třídy. Nedělitelné třídy se dá dosáhnout pomocí přidání přesně stanoveného modifikátoru do deklarace třídy, kterou chceme označit za neděditelnou.

Dědičnost byla poprvé použita roku 1968 v jazyce Simula.

Zapouzdření

Zapouzdření zabraňuje objektu přistupovat k vnitřní struktuře jiných objektů, které s ním komunikují. To znamená, že objekt je schopen pracovat se zprávami a úkoly, které dostává od ostatních objektů, ale jeho metody a atributy jsou před nimi zcela skryty a jsou použitelné výhradně pro zapouzdřený objekt. Hlavní výhodou zapouzdření je možnost měnit vnitřní strukturu objektu, pokud necháme viditelné rozhraní stejné a nepoškodíme tím celý systém (například změníme důležitý výpočet, který je nutný pro výsledek). Zapouzdřený objekt je často označován jako "black box". Rozhraní zapouzdřené třídy rozčlení třídy na public (veřejné a dostupné pro další objekty) a private (vnitřní nedostupné), což má za následek zpřehlednění a ulehčení práce se zdrojovým kódem. Každý objekt poté může abstraktně komunikovat se svým okolím, aniž by věděl, jakým způsobem jeho okolí pracuje.

4 Jazyk C#

Tento multi-paradigmatický jazyk byl vyvinut firmou Microsoft společně s .Net Framework a poprvé se objevuje v roce 2000. Jazyk C# je stejně jako většina populárních jazyků dnešní doby programovací jazyk plně objektově orientovaný a vznikl na základě obohacení jazyka C++ specifickými objektovými rysy s přihlédnutím na tehdejší aktualizace v jazyce Java (například poměrně prostou dědičnost s možnou násobnou implementací rozhraní) . V dnešní době je jedním z nejvíce používaných programovacích jazyků společně s jazyky Java, Delphi a stále ještě populárním C++. Přestože je jazyk C# plně objektově orientován, často se označuje za jazyk smíšený, protože je schopen kompilovat i programy, které nebyly navrženy jako objektové, ale které byly vytvořeny pro jazyk C a C++.

Jazyk C# také obsahuje nativní podporu komponentového programování, podporuje atributové programování a je takzvaně case sensitive (což znamená že je schopen rozlišovat velká a malá písmena v kódu).

Momentální Verze jazyka C# je C# 5.0, který byl uveden na trh v roce 2012 společně s .NET 4.5. Vedoucím vývoje architektury C# je Anders Hejlsberg.

C# je standardizován standardem ECMA Standard 334 4th edition z 2006. (ECMA, 2012) (Microsoft, msdn.microsoft.com, 2013) (Marek, 2007)

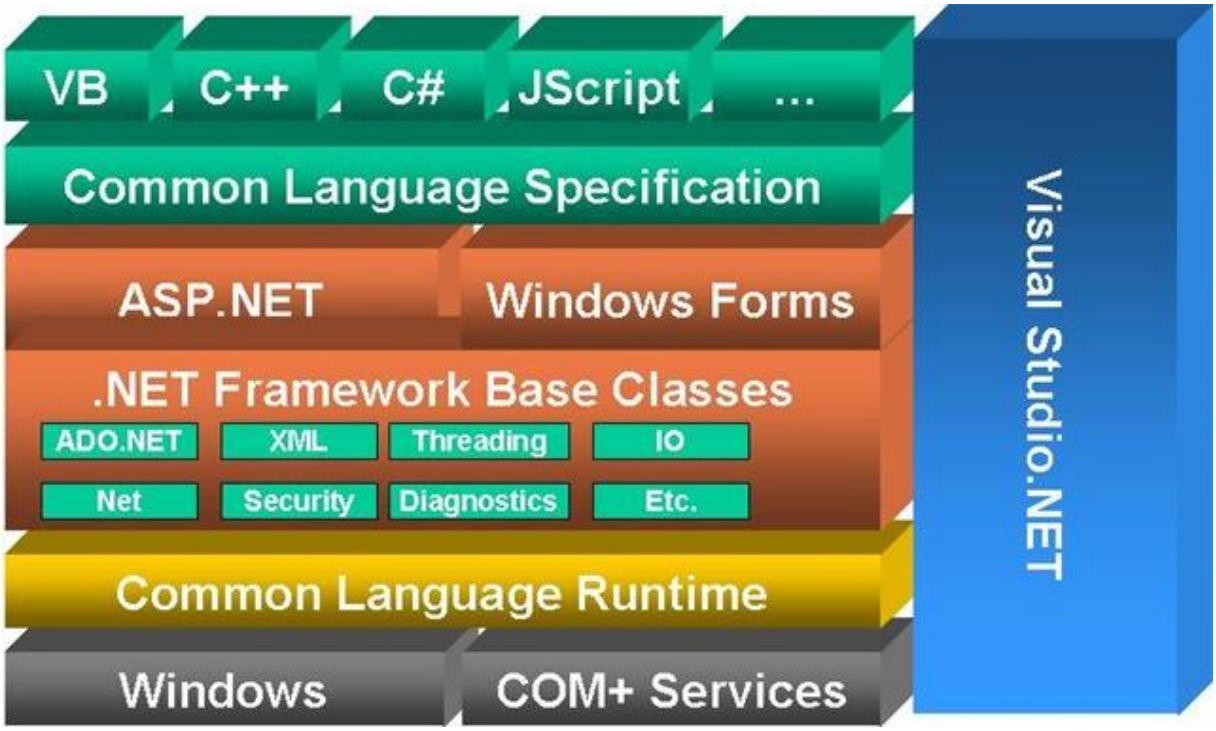
Správa paměti C#

Co se správy paměti týče, je plně automatická a plně obnovuje paměť, kterou program již nevyužívá a využívat se ani nechystá, čímž dosahuje efektivnější správy paměti a především to, že programátor se nemusí sám starat o správu paměti. C# na destrukci objektů využívá tak zvaný garbage collector, pomocí kterého identifikuje bloky paměti, se kterými již program nepracuje a v budoucnu pracovat nebude. Tyto paměťové bloky garbage collector poté uvolní a zpřístupní k dalšímu použití.

Značná část těchto vlastností vychází již ze specifikace platformy .Net, do níž je C# integrován. (Marek, 2007)(Merunka, 2006) (Microsoft, msdn.microsoft.com, 2013)

4.1 .Net Framework

.Net Framework je vlastně platforma sloužící pro podporu software. .Net je poté název zastřešující soubor v softwarových technologií tvořící celou platformu. Jádro platformy .Net Framework pracuje na principech objektového programování a nepředepisuje nutnost používat žádný specifický programovací jazyk, ale umožňuje využívat své základní služby velkému počtu programovacích jazyků. Z toho vyplývá, že .Net Framework automaticky podporuje vlastnosti objektového programování jako: třídy, metody, dědičnost, polymorfizmus, atributy apod.. Výhodou tohoto snažení je nižší vázání k programovacímu jazyku, ale vyšší nutnost používání správných objektových komponentů. (Marek, 2007) (Microsoft, msdn.microsoft.com, 2013)



Obrázek 6: Architektura .NET ((TechieGeekx .Net Architecture)*)*

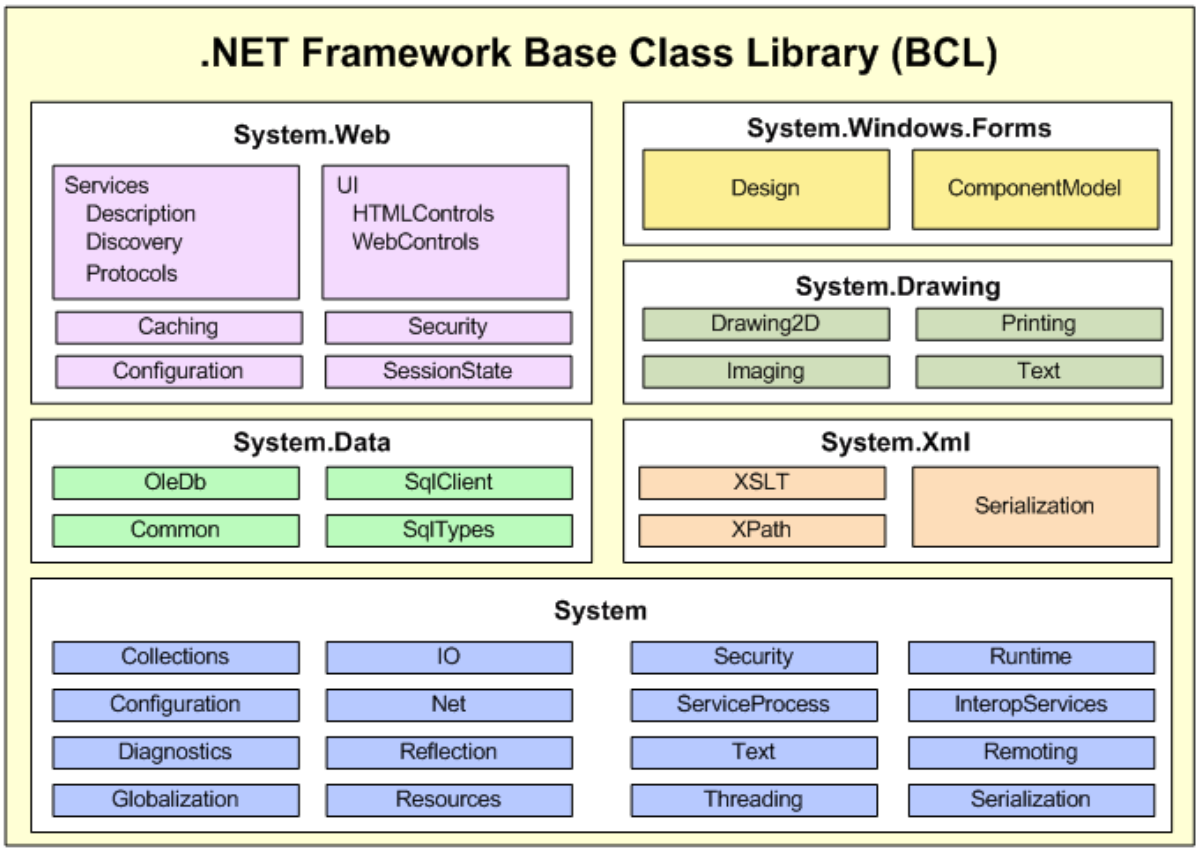
4.1.1 Architektura .NET

Co se architektury .Net Framework týče, základní rutinou realizující chod programu je Common Language Runtime, přičemž Common Language Runtime realizuje celou infrastrukturu, na které je .Net Framework vybudován. Všechny programy využívající .Net jsou spouštěny pod dohledem Common Language Runtime, což jim garantuje určitou bezpečnost a vlastnosti chování, co se zprávy paměti týče. V Common Language Runtime

jednotlivé zdrojové kódy programovacího jazyka nejsou kompilovány přímo do proveditelného nativního kódu, nýbrž do vlastního intermediárního jazyka nazývaného Microsoft Intermediate Language. Microsoft Inetermediate Language je procesově nezávislý jazyk, který se v některých ohledech vypadá jako výrazně vyspělejší verze assembleru (je schopen volat virtuální metody, pracovat s objekty a podobně). Z důvodu snahy .Net podporovat co největší počet programovacích jazyků Common Language Runtime se snaží o maximální podporu obecného typového systému (anglicky Common Type System) pro programovací jazyky. Common Language Runtime je vlastně firmou Microsoft vytvořená interpretace Common Language Infrastructure pro .Net Framework. Momentální verze Common Language Infrastructure byla schválena organizací ECMA v roce 2012 (Standard ECMA-335 6th edition). (Marek, 2007) (Microsoft, msdn.microsoft.com, 2013) (Čajda, 2009) (ECMA, 2012)

Common Type System definuje styl, kterým jsou typy spravovány, deklarovány a používány pomocí Common Language Runtime, přičemž je jednou z hlavních součástí podpory jazykové integrace v Common Language Runtime. Jeho hlavní funkce je tvorba určitého rámce, který se snaží posílit typovou bezpečnost, jazykovou integraci mezi jednotlivými programovacími jazyky a zlepšuje výkon provádění kódu. Dále poskytuje objektově orientovaný model a určuje specifikace, pomocí kterých programovací jazyky zajišťují vzájemné komunikace objektů, které jsou napsány v odlišných programovacích jazycích. V neposlední řadě také obsahuje knihovnu s základními datovými typy, kterými jsou například Char, Byte a podobně. (Microsoft, msdn.microsoft.com, 2013) (Marek, 2007)

Druhou základní částí .Net Framework je Framework Class Library. Její základní knihovnou je Base Class Library. Tato knihovna je dostupná všem programovacím jazykům, které využívají .Net Framework a obsahuje základní funkce. Mezi typické funkce můžeme uvést možnost číst, možnost psát, XML, různé schopnosti interakce s databázemi, renderování grafiky a podobně. Další důležitou sadou knihoven jsou knihovny usnadňující uživatelské rozhraní, mezi které například patří knihovny, snažící se usnadnit tvorbu webových aplikací (TechieGeekx .Net Architecture) (Marek, 2007)



Obrázek 7: Ukázka části .NET Base Class Library (wordpress)

Common Language Specification, je vlastně určitá sada pravidel a postupů, které umožňují vzájemnou spolupráci a kompatibilitu dvou a více jazyků, které jsou podporovány platformou .Net. Jazyk, který je podporován Common Language Specification, může používat knihovny, které byly vytvořeny v jiných programovacích jazycích podporujících Common Language Specification, například C# a C++. (TechieGeekx .Net Architecture)

.Net však není jedinou implementací Common Language Infrastructure. Existují i open source implementace, mezi které například patří Mono vedené firmou Xamarin. Mono v nynější verzi zcela podporuje veškeré funkce .Net s výjimkou Windows Presentation Foundation, Windows Workflow Foundation a pouze omezeně podporuje Windows Communication Foundation a ASP.NET 4.5 async stack. Výhodou je, že Mono se na rozdíl od .Net snaží zaměřit i na další platformy, jako je android a linux. V současné době již podporuje C# 5 a Common Language runtime 4.5. (Xamarin, 2011)

4.1.2 Typový systém jazyka C#

Jazyk C# využívá typový systém CTS z rozhraní .Net, který je podobně jako C++, Java, Pascal a pod. statický datový systém. Jeho typový systém se dále dělí na hodnotové a referenční typové systémy.

Jednotlivé hodnotové typy jsou nenáročné struktury v podobě znaku, čísla apod. Jinými slovy obsahují výhradně hodnotu proměnné, bez doplňkových informací. Program s nimi pracuje přímo tím, že ukládá jejich instance na svůj zásobník. V programu se často vyskytuje veliké množství hodnotových proměnných a program s nimi musí pracovat co nejrychleji. Dále je možno uvést, že hodnotové proměnné se skládají ze tří kategorií, a to základní typy, struktury a v neposlední řadě výpočtové typy. Do základních typů se zahrnují celočíselné datové typy, mezi které patří například int, short, long, byte apod. v bitovém rozmezí od 8 - 64 bitů a dále reálné datové typy jako double, float (starají se o desetinná čísla) a decimal v 128 bitovém rozmezí. Důležité je také uvést typ bool pro hodnoty true a false. Co se struktur týče, jedná se o hodnotový vstup definovaný a vytvořený uživatelem. Struktury se umisťují přímo na zásobník, i přes svoji podobu se třídami nepodporují dědičnost a nedají se využít jako základ pro nový typ. Jako poslední výpočtové typy jsou zpravidla používány pro tvorbu celočíselných konstant, z důvodu možnosti přiřazení každé položce výpočtového typu jejich hodnoty. (Marek, 2007) (Devbook.cz)

Referenční typy naopak neuchovávají hodnotu po vzoru hodnotových typů, namísto toho odkazují na specifické místo v paměti počítače, kde je instance uložena. Tato instance je nazývána objektem a její typy jsou: objekt (System.Object), string (System.String) a nové reference se dají využívat v deklaracích třída (class), Pole (array), rozhraní (interface) a delegát (delegate).

Ještě je důležité zmínit tak zvané nullable typy. Pro deklaraci nullable typů použijeme předdefinovanou třídu Nullable <T>, která je schopna uchovat hodnotu referenční, hodnotovou a i null. (Marek, 2007)

4.1.3 Preprocesor C# a jeho direktivy

C# umožňuje po vzoru C a C++ ovládat předzpracování zdrojového kódu za pomoci několika direktiv. Jednotlivé direktivy preprocesoru vždy začínají znakem #, po kterém poté následuje identifikátor názvu námi používané direktivy (například #if). Kompilátor C# však nemá

separátní preprocesor a neumožňuje tvorbu maker jako v C a C++. Toto rozhodnutí se jeví jako efektivní i z důvodu, že se zabrání tvorbě zbytečně složitých částí kódu. Jednotlivé preprocesorové direktivy můžeme rozdělit na:

podmínkové, ve kterých na základě podmínky můžeme přidat nebo odebrat část programu. Jejich zástupci jsou : #if, #else#, #elif, #endif, #define a #undef, různé

chybové a upozorňovací, které můžou složit k vypsání výstrahy, ukrytí části textu před debuggerem, nebo mohou obnovit, či potlačit určité varování compileru. Jejich typičtí zástupci jsou **:** #warning, #error, #line, #pragma, #pragma warning, #pragma checksum.

regionální, umožňují například označit část zdrojového kódu jménem. Jejich typičtí zástupci jsou: #region, #endregion.

Všechny tyto direktivy jsou podporovány od rozhraní .NET 1.1.

(Biek, 2008) (Marek, 2007) (Microsoft, msdn.microsoft.com, 2013) (Patnayak, 2012)

4.1.4 Knihovny a jmenné typy

C# je ve většině případů použit s implementací Common Language Infrastructure, ze které čerpá jednotlivé knihovny. Jednotlivé typy obdobného druhu jsou sdružovány do jmenných prostorů (namespace), které mohou obsahovat i podřízené jmenné prostory. Používání jednotlivých jmenných prostorů programu oznamujeme za pomoci příkazu using (například using System.Linq;).¨

V prostředí .NET framework jednotlivé jmenné prostory obsahuje knihovna Class Library, která umožňuje přístup k systémovým funkcím, pomocí kterých se vytvářejí nové aplikace. .NET Framework class library proto obsahuje třídy, interfacy apod., které jsou zahrnuty v .NET Framework SDK.

4.1.5 XML dokumentace C#

Jazyk C# umožňuje kromě klasické dokumentace v kódu i vytvoření XML (eXtensible Markup Language) dokumentace k třídám, které jsme vytvořili. Běžné komentáře v kódu se značí pomocí znaku //, tento znak umožňuje komentovat pouze 1 řádek kódu. Pokud potřebujeme víceřádkový komentář, použijeme znak /* ..... */. Ke generaci XML

dokumentace k našim třídám používá C# kompilátor při sestavování parametr /doc. Z důvodu, že XML komentáře nejsou metadata, v kompilovaném sestavení nejsou (zpravidla) uvedeny a není možno k nim přistupovat pomocí reflexe. Mezi tagy, které XML používá patří: c, code, example, exeption, include, list, para, param, paramref, permission, remarks, returns, see, seealso, summary a value Důležité je zmínit, že některé atributy značek include, exception, param, paramref, permission, see a seealso se při kompilování kontrolují, přičemž u značek param a paramref je kontrolována existence jejich parametru. Dále u exception, permission, see a seealso kompilátor kontroluje, zda typ, na který referují, vůbec existuje (k tomuto slouží atribut cref). Jako poslední nám zbývá značka include, sloužící k ověření existence námi referencovaného XML dokumentu. Kompilátor nám v případech neshody nehlásí chyby (error), ale varování (warning). (Puš, 2004) (Microsoft, msdn.microsoft.com, 2013)

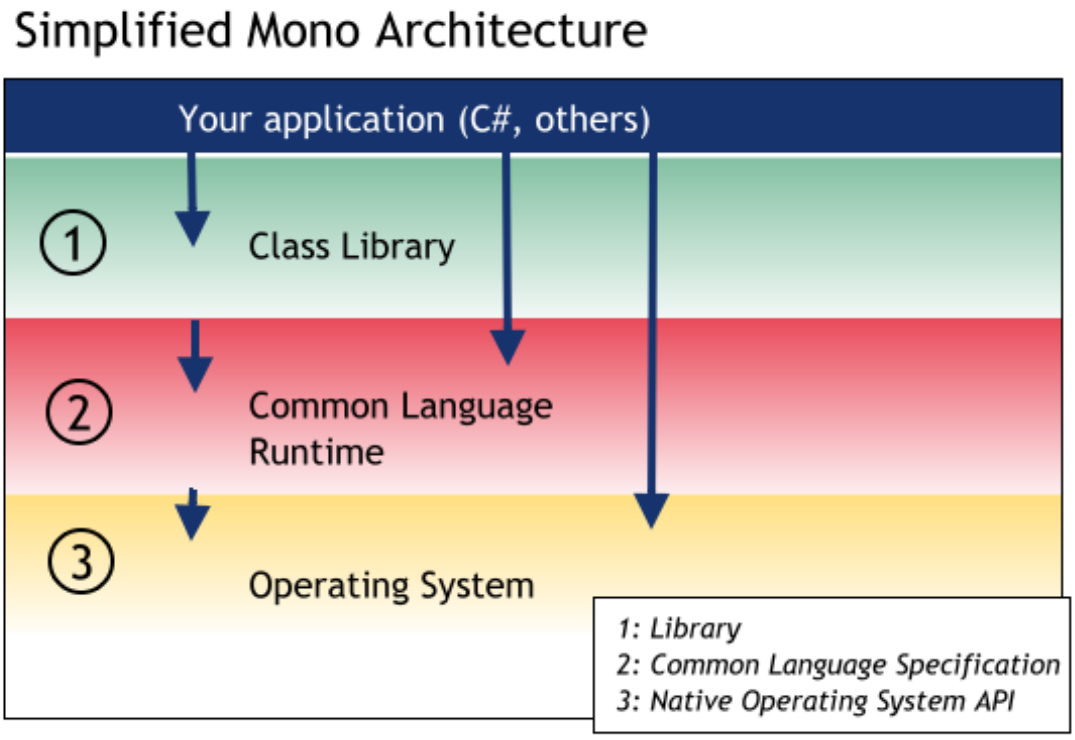
4.2 Mono

Hlavním účelem platformy mono je umožnění tvorby aplikací, které by fungovaly na více platformách (především Linuxu) pomocí .Net Framework a jedná se o otevřený software (open source). Implementace Mono byla původně založena firmou Ximian, která však byla roku 2003 koupena firmou Novel. Od roku 2011 pečuje o vývoj platformy Mono firma Xamarin. Mono jakožto implementace .Net také vychází ze standardů ECMA pro C# a Common Language Runtime.

Hlavními komponenty platformy mono jsou:

- **kompilátor jazyka C#** - obsahuje kompilátor Jazyka C# podle ECMA (1.0 -4.0)
- **Base Class Library** - Soubor knihoven, který je kompatibilní s .NET Framework
- **Mono Class Library** - Knihovny, které neobsahuje .NET Framework od společnosti Microsoft (Gtk+, OpenGL, různé Zip soubory a podobně).
- **Mono Runtime** - Mono Runtime je virtuální stroj zavádějící Common Language Infrastructure a další důležité systémové procesy jako garbage collector, načítač knihoven a podobně. Mono Runtime podporuje celou řadu operačních systémů, mezi které patří: MS Windows, Linux, Mac OS, iPhone OS, BDS, Sun Solaris, ale i konzole Play station 3 a Nintendo Wii.

(Xamarin, 2011) (Xamarin, Xamarin Guide, 2014)



Obrázek 8: Zjednodušená Architektura platformy Mono (Xamarin, 2011)

4.2.1 Mono Class Libraries

Class libraries obsahují celou řadu knihoven a je plně kompatibilní s implementací .NET od Microsoftu. Po vzoru každé implementace .NET je hlavní knihovnou námi již zmíněná Base Class Library. Mezi další knihovny patří .NET Compatibility Class Libraries a Mono Class Library. Přestože jsou Class Libraries napsány z hlavní části v jazyce C#, ale dají se použít pro každý jazyk podporovaný Common Language Specification. Class library je po vzoru .NET hierarchicky strukturována do jmenných prostorů, přičemž stejně jako ve většině implementací .NET je nejvýše postaveným jmenným prostorem System. (Xamarin, 2011)

4.3 ECMA specifikace C# a Common Language Infrastructure

Jazyk C# a Commona Language Infrastructure jsou standardizovány neziskovou privátní organizací European Computer Manufacturers Association (ECMA). ECMA byla založena 17 června 1961 v Ženevě, kde má sídlo dodnes, za účelem tvorby celosvětových standardů pro spotřebitelskou elektroniku a informační a komunikační technologie. Kromě standardů ECMA také vytváří a uchovává reporty.

ECMA zajišťuje standardy pro: programovací jazyky (C#, C++ apod.), ECMA skript, obchodní komunikace (Business Communications), Near Field Communications (standardy pro bezdrátové komunikace mezi zařízeními na malé vzdálenosti), vysokorychlostní bezdrátové komunikace (High Rate Wireless Communications), bezpečnost produktů (různé standardy a regulace pro bezpečný provoz produktů), koncepty posuzující dopad na životní prostředí (Environmental Design Considerations), akustiku (posouzení zvukového dopadu technologií na uživatele a jeho prostředí), elektromagnetickou kompatibilitu, optické úložiště, Universal 3D open file format, Office Open XML Formats a Open XML PaperSpecification. (ECMA, ecma-international.org, 2014)

Standard ECMA-334 C# Language Specification

Standard ECMA 334 se snaží o specifikaci formy a prezentace programů, které jsou tvořeny v jazyce C#. Dále se také snaží interpretovat programy napsané pomocí jazyka C#. Specifikace Jazyka C# je momentálně ve 4 edici a to z roku 2006.

Standard ECMA 334 specifikuje:

- syntaxi jazyka C#
- jaké omezení a limity musí být zavedeny do implementace C#
- způsob presentace programů napsaných v jazyce C#
- sadu sémantických pravidel, podle kterých by měli interpreti jazyka C# pracovat

(ECMA, Standard ECMA-334 , 2006)

Standard ECMA - 355 Common Language Infrastructure Specification

Standard ECMA 355 definuje Common Language Infrastructure, za jejíž pomoci je možné spustit aplikace, které byly vytvořené pomocí různých moderních programovacích jazyků (C#, F# a podobně) v různých systémových prostředích bez nutnosti jejich úpravy a změny v důsledku změny systémového prostředí. Jednou ze snah standardu Common Language Infrastructure je brát na vědomí unikátní vlastnosti a charakteristiky všech podporovaných

prostředí, a tím zjednodušit práci tvůrcům aplikací. Standard 355 je momentálně v 6. edici z roku 2012.

Mezi části tohoto standardu patří:

- koncepty a architektura - Tato část standardu 355 podrobně popisuje a rozebírá architekturu a koncept Common Language Infrastructure a obsahuje popis metadat. Dále se tato část standardu zabývá Common Language Specification, Common Type System a Virtual Execution System.
- definice metadat a semantiky - V této části jsou popsány jednotlivé části metadat. Jejich semantika, logický obsah a jejich fyzické uspořádání.
- popis sady instrukcí pro Common Intermediate Language
- Debug Interchange Format - Popisuje postup při výměně informací o odstraňování chyb programu mezi uživateli a Common Language Infrastructure producenty.
- profily a knihovny - Tato část se zabývá popisem jednotlivých knihoven využívaných Common Language Infrastructure.
- různé doplňky - v této části se nacházejí vzorové programy napsané v Common Intermediate Language Assembly Language, dále informace o implementaci asembleru, strojově čitelný popis instrukční sady Common Intermediate Language a podobně.

(ECMA, Standard ECMA-335, 2012)

5 Microsoft Visual Studio

Vývojové prostředí Microsoft Visual studio je vlastně ucelená sada určitých na komponentách založených vývojových nástrojů a různých dalších technologií podporující vytváření vysoce výkonných aplikací, a to jak v řízeném kódu, tak i ve strojovém kódu a v nejnovější verzi využívá .NET Framework 4.5.1. Mezi aplikace, pro jejichž vývoje je určeno, patří například konzolové aplikace, aplikace s grafickým rozhraním, webové aplikace, Windows Forms apod. První verze Visual studia byla vydána Roku 1995

Kromě hlavních jazyků jako C#, C++, Javascript, Visual Basic a Visual F# Visual studio podporuje i programovací jazyky jako Python, Oxygene, M a pod..

Verze Visual studia	Verze .Net Framework	Rok Vydání
Visual Studio	X	1995
Visual Studio 97	X	1997
Visual Studio 6.0	X	1998
Visual Studio .Net	1.0	2002
Visual Studio .NET 2003	1.0	2003
Visual Studio 2005	2.0, 3.0	2005
Visual Studio 2008	2.0, 3.0, 3.5	2007
Visual Studio 2010	2.0, 3.0, 3.5, 4.0	2010
Visual Studio 2012	2.0, 3.0, 3.5, 4.0, 4.5	2012
Visual Studio 2013	2.0, 3.0, 3.5, 4.0, 4.5, 4.5.1	2013

Tabulka 1: Přehled verzí Visual Studia od jeho vydání roku 1995 (vytvořeno vlastní činností na základě (Microsoft, msdn.microsoft.com, 2013)*)*

Částmi dnešních verzí MS Visual studia: jsou sady: Microsoft Visual C++, Microsoft Visual C#, Visual F#, Microsoft Visual Basic, Microsoft Visual Web Developer (podpora html5, javascript, CSS3 a JQuery) a Team Foundation Server. Jazyk C#, na který se dále zaměříme, byl poprvé použit ve verzi Visual Stuido .Net. (Microsoft, http://www.visualstudio.com/, 2014) (Microsoft, msdn.microsoft.com, 2013)

5.1 Programovací Jazyky ve Visual Studiu

Visual C++

Visual C++ je flexibilní sada umožňující tvorbu nových aplikací pomocí jazyků C a C++. Pro jazyk C++ udržuje specifikaci ANSI C++ a skládá se ze:

- Visual Studio development environment - což je prostředí, ve kterém se tvoří jednotlivé aplikace
- Visual C++compiler tools - Kompilátor podporuje zároveň vývoj směřovaný na .NET Common Language Runtime, tak současně i vývoj v nativním kódu.
- Libraries - Jednotlivé knihovny, které Visual C++ využívá a patří mezi ně například: C Runtime Library, Standard C++ Library, Active Template Library a podobně. (Microsoft, http://www.visualstudio.com/, 2014) (Microsoft, msdn.microsoft.com, 2013)

Visual C#

Visual C# jak již název napovídá je realizace programovacího jazyka C# společností Microsoft. Visual C# má v sobě zabudován editor kódu, kompilátor, návrháře, průvodce kódem, různé šablony projektů ladící program a různé další nástroje, které Visual Studio umožňuje využívat. Visual C# přistupuje k třídám a většině služeb operačního systému pomocí knihoven .NET Framework. Důležité je také zmínit, že překladač Visual C# je součástí .NET Framework a je k dispozici separátně. Visual Studio 2013 momentálně podporuje nejnovější verzi jazyka C# 5.0. (Microsoft, msdn.microsoft.com, 2013)

Javascript

Visual studio 2013 obsahuje editor jazyka Javascript s podporou technologie IntelliSense, pomocí které je možno psát efektivnější kód, s nižším počtem chyb a lepším přístupem k informacím. Technologie IntelliSense má snahu o zjednodušení a zefektivnění hledání informací o jednotlivých členech, vkládání jazykových prvků do zdrojového kódu programu, snahu o udržení kontextu bez nutnosti opuštění editoru a podporuje vlastní dokumentaci pomocí XML. (Microsoft, msdn.microsoft.com, 2013)

Visual F#

Visual F# je editor multi-paradigmatického jazyka, který se snaží o spojení funkcionálního programování (například jazyk Haskell) a imperativního objektově orientovaného

programování pro .NET. V základu syntaxe jazyka F# vychází z jazyka ML, přejímá však některé konstrukce i z jiných jazyků jako C#, OCaml, Haskell a podobně. Systém knihoven Visual F# vychází z .Net a hlavní knihona je FSharp.Core.dll. Knihovny Visual F# také podporují různé paralelní výpočty a paralelní pracovní postupy (dokonce i asynchronní postupy). Stejně jako všechny jazyky platformy .NET Framework i F# zcela bezproblémově pracuje se všemi podporovanými jazyky (jako například C#).

Visual studio také podporuje interaktivní skriptování jazyka F#, což má za následek možnost testování kódu v průběhu jeho tvorby.

Poslední verze jazyka je F# 3.1, která je implementována ve Visual studio 2013. (Microsoft, msdn.microsoft.com, 2013)

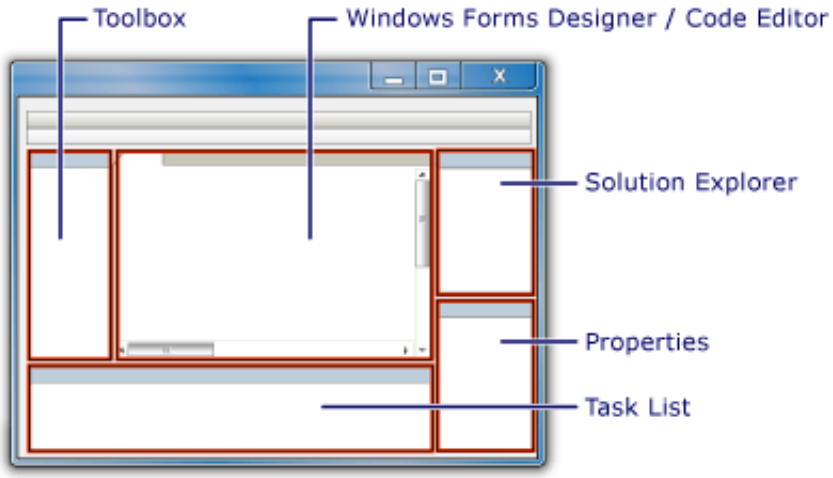
Visual Basic

Visual Basic je multi-paradigmatický a typově bezpečný programovací jazyk, který je schopen tvorby objektově orientovaných aplikací. Přestože byl Visual Basic navrhnut jako přístupnější programovací jazyk, disponuje velice dobrou flexibilitou a je schopen tvorby aplikací jak pro MS Windows, tak i pro různá mobilní zařízení a web. Jako všechny implementace programovacích jazyků ve Visual Studiu je i Visual Basic zaměřen na Microsoft .Net Framework. (Microsoft, http://www.visualstudio.com/, 2014) (Microsoft, msdn.microsoft.com, 2013)

5.2 Visual C# integrated development enviroment (IDE)

Integrated development enviroment Visual C# je na první pohled poměrně klasické uživatelské rozhraní obsahující základní nástroje, lišty, ikony a přehled jednotlivých hlavních nabídek nástrojů situovaných na horní liště. Prostředí sdílí velkou část nástrojů se všemi ostatními prostředí programovacích jazyků Visual Studia. Některé části však není možno efektivně sdílet a jsou unikátní. Přestože od své původní verze Visual C# poměrně změnil vzhled a získal rozsáhlou řadou nových nástrojů, orientace v Prostředí zůstává obdobná (například panel nástrojů (Toolbox) je stále na levé straně již od roku 2002 kdy byl Visual C# poprvé implementován). Moderní verze umožňují poměrně rozsáhlé možnosti nastavení prostředí, což umožňuje uživateli zjednodušení práce ve Visual C#.

Nejdůležitější sady nástrojů jsou dostupné z hlavní lišty, na které jsou nabídky file, edit, veiw, project, build, debug, team, tools, test, window a help. Mezi nejdůležitější nástroje Visual C# se dá zařadit:



Obrázek 9: Prostředí Visual C# (Microsoft, Introduction to the IDE (Visual C#), 2008)

- editor používaný pro psaní zdrojového kódu v jazyce C#
- debuger dostupný přímo ikonou na hlavní liště (záleží na nastavení), nebo pomocí nabídky debug, soužící k testování funkčnosti kódu
- kompilátor převádějící námi napsaný zdrojový kód programu na spustitelný program
- sady nástrojů, sloužící pro lepší práci s programem a sloužící ke grafické tvorbě programu (jednoduchá implementace textu, tlačítek a podobně)
- průzkumník řešení (Solution Explorer), který má na starost hierarchické zobrazení projektů a řešení, tvorbu, zobrazování a umožnění práce s vnějšími soubory a vedlejšími soubory, tvorbu a řízení zabudovaných souborů a dalších nástrojů.
- návrhář projektu (Project Designer), sloužící pro konfigurace možností a nastavení kompilátoru, nastavení jednotlivých cest a dalších nastavení
- okno zobrazující vlastnosti (Properties Window), toto okno slouží převážně pro kontrolu uživatelského prostředí, ale také zobrazuje vlastnosti a události prostředí
- zobrazení tříd (Class View), nám umožňuje orientaci a navigaci v zdrojovém kódu podle jednotlivých typů
- prohlížeč objektů (Objeckt Browser), zjednodušuje a umožňuje prohlížení jednotlivých tříd a metod dostupných v knihovnách .NET.

- okno ukazující výstup naší aplikace (task list), který je ve výchozím nastavení situován ve spodní části obrazovky a určuje nám výstup našeho programu, popřípadě upozorňuje na chyby, kterých jsme se dopustili.
  (Microsoft, Introduction to the IDE (Visual C#), 2008) (Microsoft, msdn.microsoft.com, 2013)

6 Xamarin Studio

Xamarin Studio je nové moderní vývojové prostředí využívající jazyk C#, zaměřené převážně na tvorbu aplikací pro iOS, MacOS a Android. Xamarin Studio bylo vytvořeno s myšlenkou umožnit multi-platformní tvorbu aplikací zachycující celý vývojový cyklus programu, a to od začátku jeho vývoje až po jeho dokončení a publikaci. Krom podpory C# Xamarin Studio také podporuje (podobně jako Visual Studio) F#, pro všechny své platformy. (Paul, 2013) (Xamarin, Xamarin Guide, 2014)

Xamarin studio je z velké části vytvořeno na základě MonoDevelop, což jak již bylo řečeno, je prostředí IDE vytvořené na podporu vývoje pomocí .NET s platformou Mono převážně zaměřené na operační systém Linux. MonoDevelop byl založen na podpoře Gtk+ a podporoval jazyky C, C++, C#, F#, Python, Oxygene, CIL, Vala, Boo, Visual Basic a Java. Z důvodu rozhraní Gtk+, se dá říci, že MonoDevelop byl poměrně úzce spjat s platformou Gnome. Přestože Xamarin Studio navazuje na MonoDevelop, snaží se o modernější přístup v rozložení prostředí a ustupuje od panelů nástrojů plných ikon směrem k na první pohled jednoduššímu a lépe přehlednému modernímu prostředí. Důležité je zmínit, že Xamarin stále využívá Gtk+ pro multi-platformní kompatibilitu. V dnešní době je možno MonoDvelop ve své nejnovější verzi sehnat jako součást Xamarin Studia.(Paul, 2013)

Xamarin studio je součástí platformy Xamarin, která se skládá ze čtyř částí a to:

- **Mono .NET framework**
- **Compiler**, který je závislý na platformě (například integrované .NET aplikace a runtime pro android)
- **Integrated development enviroment** (Samotné Xamarin studio, nebo integrace Xamarinu pro MS Visual Studio)
- **Jazyk C#**

(Xamarin, Understanding the Xamarin Mobile Platform, 2014)

Nejstarší podporované verze (údaje z roku 2014, Xamarin 4.2.5, stejné i pro Xamarin 5.0) Androidu a iOS jsou:

- **Android 4,0**
- **iOS 6,1**

Xamarin také umožňuje psát programy pro **Windows Phone 8**, ale pro tyto programy je nutno užít jeho implementace s Visual Studiem.

Kompilace (překlad) kódu

Samotná kompilace kódu se výrazně liší u jednotlivých platforem, přičemž iOS a android neumožňují některé funkce podporované .NET. Tyto funkce jsou vypsány v Xamarin.iOS Limitations a Xamarin.Android Limitations. Aplikace pro iOS jsou kompilovány způsobem ahead-of-time (překlad v době instalace) a jsou překládány do jazyka symbolických adres ARM. V případě Androidu a Windows Phone je C# kompilován do Intermediate language, přičemž pro android jsou přidány MonoVM a JIT´ing. (Xamarin, Understanding the Xamarin Mobile Platform, 2014) (Xamarin, Xamarin Guide, 2014)

Software development kit

Jednotlivé sady nástrojů pro tvorbu aplikací jsou pro jednotlivé podporované platformy odlišné. Xamarin poté pracuje s vybranými sadami nástrojů pro cílovou platformu. Xamarin.Android poté přistupuje k sadám nástrojů Google Android jako k jmenným typům a je k nim možno přistupovat pomocí příkazů ve stylu using Android.app;. Podobně se přistupuje i u Xamarin.iOS, k sadám nástrojů Apple CocoaTouch. Jednotlivé sady nástrojů je nutné instalovat samostatně (nejsou přímo zabudovány v Xamarin stuidu) a například u platformy android je nutné stáhnout také Android SDK Manager, který umožní stažení jednotlivých nástrojů a rozhraní pro programování aplikací (aplication programming interfeace). (Xamarin, Understanding the Xamarin Mobile Platform, 2014) (Xamarin, Xamarin Guide, 2014)

Rozdíl je u Windows Phone, které nejsou součásti platformy Xamarin a aplikace pro ně tvořené za pomoci jazyka C# jsou implicitně dostupné. (Xamarin, Understanding the Xamarin Mobile Platform, 2014) (Xamarin, Xamarin Guide, 2014)

Xamarin Test Cloud

Xamarin Test Cloud umožňuje externě a automatizovaně otestování programů vyvinutých na celou řadu mobilní zařízení pomocí User Interfeace Acceptance Testing. Tato utilita byla zařazena z důvodu, že nejlepší test aplikace je vždy na příslušném zařízení. Celý proces probíhá taky, že je aplikace podrobena testu přístupnosti uživatelského rozhraní (User Interfeace Acceptance Testing (většinou ve zkratce API)), ve kterém je aplikace otestována na maximálním počtu zařízení. Tento styl testování je velmi nákladný na realizaci samotným vývojářem, či jeho společností. (Xamarin, Xamarin Guide, 2014)

Samotný Test Cloud umožňuje dvě možnosti otestování aplikace a to pomocí vytvoření vlastního testu anebo pomocí Test Cloud App Exploreru. Vlastní test je možno vytvořit za pomocí jednoho ze dvou frameworků, které Xamarin Test Cloud využívá a to: Cabalash a Xamarin.UITest.

Test Cloud App Explorer na druhou stranu automaticky a systematicky prochází zaslanou aplikaci, přičemž se snaží reagovat na její vlastnosti. Výhodou tohoto druhu testu je možnost rychle otestovat aplikací, nevýhodou je však nedostatečná hloubka otestování aplikace. Pokud je nutno odhalit složitější nedostatky programu, je lepší použít vlastní test. (Xamarin, Xamarin Guide, 2014)

6.1 Xamarin Studio integrated development enviroment (IDE)

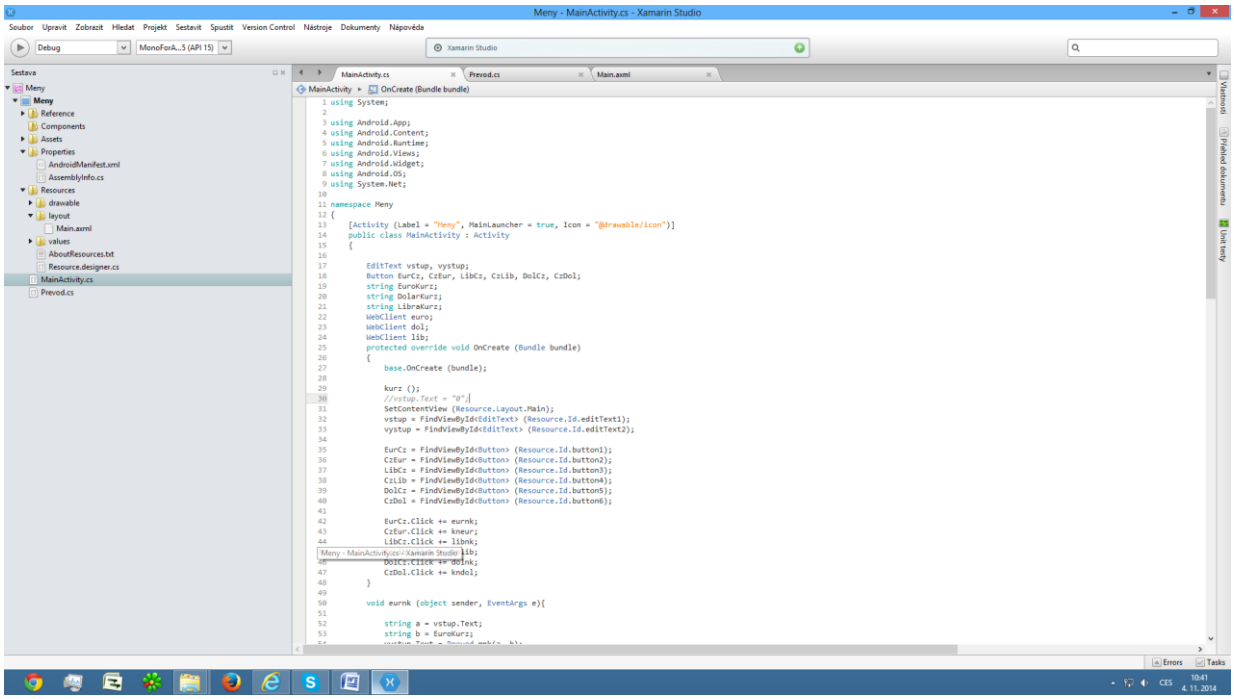
Xamarin Studio IDE je sestaveno podobně jako prostředí Visual Studio do několika sekcí, které se starají o chod aplikací, souborů, tvorby zdrojového kódu, nastavení, testování funkčnosti programu a podobně. Co se sad nástrojů týče, tak Xamarin studio je poměrně podobně MS visual studiu a na tvorbu aplikací využívá obdobné nástroje, které byly popsány u Visual Studia a umožňuje klasické vlastnosti editoru kódu jako například zvýrazňování syntaxe jazyka (důležitá slova jsou barevně odlišena), automatické doplňování kódu v průběhu jeho tvorby. (Xamarin, Xamarin Guide, 2014)

Přístup k tvorbě a řízení jednotlivých projektů je vytvořen po vzoru Microsoft Visual studia a v základu vychází ze stejné myšlenky: tvorba hlavního řešení, na které umisťuje libovolné množství dalších součásti projektu. Samotná struktura a obsah závisí na jednotlivé platformě, pro kterou je aplikace tvořena (Android, iOS, WIndows mobile). Také je nutné zmínit, že při

testování aplikace se automaticky spustí emulátor platformy (například Mono for android API 15), pro kterou je aplikace tvořena.

Oproti MS Visual Studiu je však důležité uvést, že Xamarin studio je poměrně náročnější na výkon počítače a například Debuger pracuje výrazně déle než u MS Visual Studia. Toto je z velké části způsobeno multiplatformní tvorbou (převážně spouštěním emulátoru a emulováním jiného systému).

Xamarin, jak již bylo zmíněno, také podporuje tvorbu softwaru za pomocí MS Visual Studia, a tím umožňuje Visual Studiu tvořit aplikace pro Android a iOS, přičemž tato možnost je nutná pro vývoj software pro Windows Phone. Hlavní výhodou použití Visual Studia pro tvorbu aplikací pro andoid a iOS je jednotné vývojové prostředí pro vývoj na většinu platforem (včetně Windows desctop).(Xamarin, Visual studio with xamarin, 2014)



Obrázek 10: Ukázka vývojového prostředí Xamarin Studio (vytvořeno vlastním zpracováním)

7 Tvorba vzorové aplikace

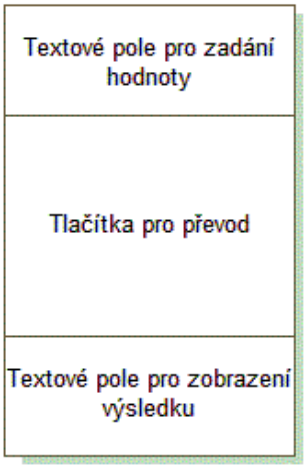
7.1 Návrh aplikace

V této části se budeme zabývat tvorbou jednoduché aplikace pro převod měn pomocí .NET a jazyka C#. Jakožto vývojové prostředí budou použity MS Visual Studio a Xamarin studio. Aplikace tvořena za pomocí MS Visual Studia bude směřována na desktopovou platformu Windows a aplikace tvořena za pomocí Xamarin studia bude směřována na mobilní platformu Android. Postup tvorby aplikace bude po rozdělen do několika částí, přičemž pokud bude postup rozdílný, jako první bude vždy použito MS Visual Studio.

Požadavky kladené na vzorovou aplikaci

V této části si rozebereme základní požadavky, které budeme mít na tvorbu a finální vzhled námi tvořené aplikace.

- Samotná aplikace bude vytvořena pro grafický interfeace zpřístupněný příslušným studiem a bude obsahovat 2 textové pole a 6 tlačítek umožňujících převod. Jedno textové pole bude pro vstup hodnot, druhé bude zobrazovat výslednou hodnotu po převodu.



Obrázek 11: Návrh prostředí aplikace (vytvořeno vlastním zpracováním)

- Do zadávacího textového pole bude možno psát pouze čísla.
- Výsledkové pole bude zablokováno pouze pro přepočty a nebude možno do něj psát.
- Pokusíme se co nejméně zasahovat do XML kódu při tvoření grafické části aplikace.
- Kurzová data budou stahovány z stránek ČNB, tak aby vždy odpovídala aktuálnímu kurzu.

7.2 Tvorba grafického designu aplikace

V této části rozebereme práci s grafickým designerem, který MS Visual Studiu a Xamarin Studiu implicitně obsahuje. Zaměření tohoto srovnání bude především na rozdíly, které nastaly při tvorbě námi určené aplikace.

7.2.1 Tvorba a umístění jednotlivých nástrojů

Na rozmístění jednotlivých tlačítek, názvů a textových polí nám v případě obou Studií postačí výhradně práce s nástrojovou sadou (Toolbox). Výhodou MS Visual studia je grafické nabídnutí automatického upravování velikostí a umístění jednotlivých nástrojů při jejich vkládání a měnění velikosti bez nutnosti úprav v Xml. Xamarin bohužel neupravuje rozmístění jednotlivých nástrojů automaticky a je nutné dopisovat jednotlivé rozmístění v XML kódů.

7.2.2 Úprava textových polí

Zadávací textové pole

Podmínkou je, aby textové pole pro hodnoty určené k převedení akceptovalo výhradně čísla a tím zabránilo chybám v převodu z důvodu zadání nesmyslných hodnot. V Xamarin Studiu je toto ošetření velice snadné, a to pomocí položky imput type a jejího nastavení na Number. U Visual Studia je však nutné tuto funkci doprogramovat. V oknu vlastností v sekci události figuruje položka KeyPress a pomocí, které lze vytvořit metodu Vstup_KeyPess. Do metody je nutno napsat následující kód:

```
private void Vstup_KeyPress(object sender, KeyPressEventArgs e)
{
char t = e.KeyChar;
if(!Char.IsDigit(t) && t !=8 && t != 46)
{
e.Handled = true;
}
```

Tato metoda se pří zmáčknutí klávesy zeptá, jestli není číslo nebo backspace (8) a Del (46). Pokud nastane tato situace, do textového pole se nic nenapíše (`e.Handled = true`). Zkratky vycházejí z konfigurace jednotlivých kláves pro C#, které jsou dostupné na stránkách Microsoftu (MSDN Keys Enumeration).

Výsledkové textové pole

Pro nastavení uzamčení textového pole určeného pro zobrazení výsledku je postup ve Visual Studiu a Xamarin Studiu identický. Samotná procedura je realizována u Visual Studia pomocí okna vlastností (Properties) kolonkou ReadOnly a jejím nastavením na True. U Xamarin Studia se dá použít namísto EditText TextWiev v panelu nástrojů. Pokud je použit EdtiText, tak se nastaví v jeho parametrech android:editable="false" (jak bylo užito ve vzorové aplikaci).

7.3 Tvorba zdrojového kódu aplikace

Přestože obě prostředí používají stejný programovací jazyk, zdrojový kód obou aplikací není totožný. V této části práce budou vyzdviženy rozdíly ve zdrojovém kódu mezi tvorbou pro android a MS Windows.

7.3.1 Knihovny

Při vytvoření nového projektu se jak v Xamarin studiu, tak i v MS Visual studiu automaticky načte většina potřebných knihoven. Oboje aplikace mají jako první knihovnu vždy knihovnu using System;, obsahující mnoho běžně používaných typů. Zbytek automaticky načtených knihoven je již rozdílný, například u Xamarin studia jsou všechny zaměřeny na android (např.: using Android.OS;).

Aplikace bude však používat službu WebClient, která není součástí automaticky načtených knihoven a bude nutné dopsat knihovnu nutné pro tento příkaz. Oboje prostředí používají knihovnu using System.Net;, která obsahuje programovací prostředí pro řadu síťových protokolů. Jejími typickými zástupci jsou například třídy WebRequest a WebResponse, ale právě také WebClient. (Microsoft, msdn.microsoft.com, 2013)

Co se týče knihoven, postup tvorby je prakticky totožný, hlavní rozdíl je načtení jiných knihoven při tvorbě aplikace, ale doplněná knihovna System.Net je používána v obou prostředích.

7.3.2 Proměnné a načtení nástrojů z grafického designu

Z důvodu přehlednosti jsou proměnné definovány na začátku programu a ne v průběhu tvorby. Hlavní proměnné nutné pro převod a stahování z internetu budou shodné pro obě prostředí.

Pro převodové proměnné je použit jednoduchý typ string, ukládající hodnoty EuroKurz, DolarKurz a LibraKurz (obsahující hodnoty eura, dolaru a libry), která bude převáděna, pokud by kurz nebyl stahován z internetu, ale zadal by se fixní kurz hodnoty by byly nastaveny například double EuroKurz = 27, což by znamenalo nutnost změny typu z textového formátu string na formát double. Tyto proměnné poslouží jak pro převod tak pro ukládání hodnot, které byly staženy z webu české národní banky.

```
string EuroKurz;
string DolarKurz;
string LibraKurz;
```

Dále v programu u jednotlivých převodových tlačítek se nachází proměnné string a; a string b;. tyto proměnné poslouží k převodu měn.

Do programu pro MS Visual Studio bude doplněna ještě proměnná char t;, která byla popsána v předchozí kapitole.

Zásadní rozdíl nastává v načtení designeru. Zatím co MS Visual Studio načítá tlačítka a textové pole automaticky a jakmile jsou dány v designeru není nutné dopisovat k nim kód. Xamarin Studio pro android vyžaduje dopsání parametrů do zdrojového kódu. Pro jednotlivá tlačítka a textové pole je nutné vytvořit následující proměnné, Přičemž EditText zastupuje textové pole a Button zastupuje tlačítka převodu:

```
EditText vstup, vystup;
Button EurCz, CzEur, LibCz, CzLib, DolCz, CzDol;
```

Dále je těmto proměnným nutné přiřadit jednotlivá tlačítka a textové pole, které byly vytvořeny v designeru, cele přiřazení se píše do části kódu pod On Create:

```
vstup = FindViewById<EditText> (Resource.Id.editText1);
vystup = FindViewById<EditText> (Resource.Id.editText2);
EurCz = FindViewById<Button> (Resource.Id.button1);
CzEur = FindViewById<Button> (Resource.Id.button2);
LibCz = FindViewById<Button> (Resource.Id.button3);
CzLib = FindViewById<Button> (Resource.Id.button4);
DolCz = FindViewById<Button> (Resource.Id.button5);
CzDol = FindViewById<Button> (Resource.Id.button6);
```

Jako poslední si nadefinujeme proměnné třídy WebClient. Tato třída poskytuje metody pro jak příjem dat, tak i jejich odeslání. Právě tyto proměnné nám budou sloužit ke stažení kurzu z internetu. (Microsoft, msdn.microsoft.com, 2013)

```
WebClient euro;
WebClient dol;
WebClient lib;
```

7.3.3 Metody sloužící pro převod pomocí tlačítek

Jako první je nutné vytvořit třídu, ve které budou realizovány převody. Postup je prakticky totožný v obou prostředích, a to pomocí příkazu create new class. Jediný rozdíl je, že Visual studio si načte několik dodatečných knihoven automaticky, zatímco Xamarin studio pouze using system.

Syntaxe jazyka C# je totožná, takže metody vytvořené v prvním prostředí je možno použít i v druhém prostředí. Pro převod je nutno vytvořit v třídě form1 (popřípadě MainActivity u Xamarin Studia) pro každé tlačítko metodu, která bude používat proměnné string a a string b. Jako vzorovou použijeme metodu převodu korun na eura.

```
private void CzknaEur_Click(object sender, EventArgs e)
{
string a = Vstup.Text;
string b = EuroKurz;
Vystup.Text = Prevod.knm(a, b);
}
```

Do proměnné string se ukládá textový řetězec, který byl napsán do vstupního textového pole. Do hodnoty string b, se ukládá hodnota příslušné měny stažená z internetu. Jako poslední tato metoda odkazuje proměnné **a** a **b** do třídy Prevod a její metody knm, přičemž následně výsledek převodu vypisuje na výstupní textové pole.

V Prostředí Xamarin je situace oproti MS Visual Studiu složitější. Přestože kód jednotlivých metod je stejný, je nutné k daným metodám přiřadit tlačítka pomocí příkazů:

```
EurCz.Click += eurnk;
CzEur.Click += kneur;
LibCz.Click += libnk;
CzLib.Click += knlib;
DolCz.Click += dolnk;
CzDol.Click += kndol;
```

Samotný převod bude zajišťován v třídě Převod. Bude použita metoda typu string, která bere proměnné **a** i **b.** Pro všech 6 tlačítek stačí pouze 2 převodové metody, protože převod z korun na jinou měnu je vždy totožný výpočet. U obou metod bude jako první figurovat parametr return, navracející hodnotu do volající metody (například CzknaEur). Dále bude konverze na textový řetězec, který bude použit na výstupním textovém poli (přestože vstupní řetězec je také string, bude nutné znovu konvertovat na string z důvodu výpočtu za pomoci double). Do tohoto textového řetězce bude napsáno **a** konvertováno na double (číselnou hodnotu pro výpočet) a dle metody ***** nebo **/ b** konvertováno na double.

```
internal static string knm(string a, string b)
{
return Convert.ToString((Convert.ToDouble(a) *
Convert.ToDouble(b)));
}
internal static string mnk(string a, string b)
{
return Convert.ToString((Convert.ToDouble(a) /
Convert.ToDouble(b)));
}
```

Postup pro tvorbu metod v třídě Převod je zcela identický pro obě vývojová prostředí.

7.3.4 Stahování kurzu z internetu

V této části bude popsána část stažení denního kurzu pro jednotlivé měny z webu České národní banky. Postup stažení kurzu bude opět téměř shodný pro obě prostředí (to znamená jak pro android tak pro Windows OS). Jako první je nutné vytvořit metodu kurz(), která bude obsahovat celý postup.

Na stahování kurzu je nutné použít třídu WebClient, pro kterou byla přidána knihovna using System.Net. a pro kterou byly vytvořeny proměnné euro, dol, lib. Jako vzor bude popsán postup pro stahování měny euro, postup tvorby pro ostatní měny je zcela identický.

Postup je, že pro euro se vytvoří nová třída WebClient. Do proměnné euro kurz se za pomoci třídy WebClient stáhne textový řetězec pomocí příkazu DownloadString (WebClient umožnuje mimo jiné například i DownloadFile nebo DownloadData) z webu České národní banky. Důležité je zmínit, že na České národní bance je nutné nastavit formát jako text. Web České národní banky vygeneruje odkaz z kurzu, který je nutno zkopírovat do kódu za ("http://www.cnb.cz/miranda2/m2/cs/financni_trhy/devizovy_trh/kurzy_devizoveho_trhu/vybrane.txt?mena=EUR&od=01.08.2014&do=31.08.2014"). V této části může nastat rozdíl, pokud je program pro android vytvářen v Xamarin studiu. Xamarin studio není schopné převést symbol desetinné čárky "," z internetu, a proto je nutné zvolit na stránkách České národní banky anglickou verzi internetového výstupu, kde symbol desetinné čárky je ".". Z odkazu je patrné, že jednotlivé kurzy jsou pevně dány na datum od začátku určitého měsíce do jeho konce (zde například srpen). Z důvodu, že cílem je udělat vždy aktuální kurz, je nutné upravit hodnoty od do na parametry od {0} do {1}, přičemž je nutno celý odkaz upravit do formátu String pomocí příkazu String.Format. Zbývá jen dosadit za odkaz do parametrů pomocí třídy DateTime, přičemž je nutné použít dnešní datum (.Today), pro první parametr je nutné dát několika denní zpoždění z důvodu, že o víkendech Česká národní banka neaktualizuje kurz. Samotné zpoždění se realizuje parametrem AddDays a z důvodu nutnosti několikadenního zpoždění se parametr nastaví na hodnotu -6. Dále je nutné dodat přesnou hodnotu data, podle webové stránky. Z důvodu, že náš cíl odděluje Datum tečkami, musí být hodnota data nastavena ToString("dd.MM.yyyy"). Pokud by byly hodnoty data odděleny lomítky, bylo by nutné napsat ToString("dd/MM/yyyy"). (Microsoft, msdn.microsoft.com, 2013) (Ing. Petr Voborník)

```
euro = new WebClient();
EuroKurz =
euro.DownloadString(String.Format("http://www.cnb.cz/miranda2/m2/cs/
financni_trhy/devizovy_trh/kurzy_devizoveho_trhu/vybrane.txt?mena=EU
R&od={0}&do={1}",DateTime.Today.AddDays(-6).ToString("dd.MM.yyyy"),
DateTime.Today.ToString("dd.MM.yyyy")));
```

Z důvodu, že webový výstup vypadá například takto: 01.08.2014|27,640 a my potřebujeme hodnoty za symbolem |. je nutné oddělit část textu pomocí příkazu Substring, přičemž text

bude brán jako symbol na pozici posledního výskytu znaku |. Ještě zbývá přičíst +1 z důvodu, že v kódu nechceme přímo znak |, ale veškerý text, který je za ním. (Ing. Petr Voborník)

```
EuroKurz = EuroKurz.Substring(EuroKurz.LastIndexOf('|') + 1);
```

Na závěr je nutné umístit metodu kurz() těsně pod inicializaci komponent. To se provede napsáním kurz(); přímo pod InitializeComponent(); v MS Visual Studiu a pod base.OnCreate (bundle); v Xamarin Studiu.

7.4 Zhodnocení a srovnání postupu tvorby

Přestože se jednalo o tvorbu prakticky stejné aplikace za použití stejného programovacího jazyka, postup tvorby byl poměrně odlišný. Obě vývojová prostředí mají poměrně stylizovaný a relativně podobný design a práce v nich není výrazně odlišná, osobně považuji MS Visual Studio za lépe zpracované a více uživatelsky přívětivé.

Dva hlavní rozdíly v průběhu tvorby nastávají při práci s grafickým designerem a při načítání jednotlivých tlačítek a textových polí pro platformu android. Tvorba pro android je náročnější než pro Windows (i z pohledu designerů není takový prostor pro manipulaci s grafikou), takže tento rozdíl se prakticky nedá odstranit.

Grafický designer by se však dal unifikovat pro obě prostředí, pomocí již zmíněné možnosti Xamarin for Visual Studio. Tím se sice neodbourá složitější kód nutný pro android, ale dá se využít jednodušší a přívětivější prostředí. V zásadě to však není nutné, protože prostředí Xamarin Studia je více méně obdobné kvality a jako hlavní výhoda je opravdu převážně odstranění nutnosti zvyku na víc prostředí.

Celkově se dá říci, že jednodušší je postup při tvorbě aplikace v MS Visual Studiu, který je zaměřený na cílovou platformu Windows. Xamarin studio je však velice kvalitní vývojové prostředí a v porovnání s Visual Studiem je na trhu poměrně krátkou dobu a prochází konstantními aktualizacemi, které mění i jeho vzhled a použití.

8 Závěr

Od počátků programování se vývoj software poměrně výrazně posunul a je jisté, že od dob sekvenčních jazyků se dnešní programování poměrně výrazně liší. Přestože objektový postup byl, jak již bylo zmíněno od roku 1960, ve větší míře se začal používat až v devadesátých letech. Právě jazyk C# je v dnešní době jedním z nejvíce používaných programovacích jazyků a dle mého názoru ještě poměrně dlouhou dobu bude.

Jedním z velkých rozdílů je samozřejmě změna cílových platforem. Dříve tvořily prim ve vývoji software stolní počítače, ale dnes jsou pomalu nejvíce populární mobilní platformy (ať již jde o mobilní telefony, či různé tablety). V popularitě mobilních platforem se odráží rapidní nárůst nových aplikací směřovaných právě na android, iOS, nebo Windows Mobile. Právě jazyk C# je velice silný prostředek pro vývoj softwaru pro jakoukoliv platformu, ať pro stolní počítače nebo mobilní zařízení.

Popularita mobilních zařízení určitě do budoucna poroste z důvodu nových možností cloud computing a celkového zaměření na poskytování různých programů přímo z internetového serveru. Toto bude umožňovat použití náročnějších aplikací z mobilních zařízení a tím nejspíš zvýší jejich růst a poptávku po software na ně zaměřených.

Zajímavá otázka je, kam půjde vývoj dále po objektovém programování. Osobně se domnívám, že objektový náhled na programování vydrží ještě dlouhou dobu, a to převážně z důvodu, že objektové programování se snaží být přímým odrazem reálného světa. V budoucnu je však možné, že vývoj půjde jiný nečekaným směrem nebo že se vymyslí model, který bude principiálně dokonalejší.

9 Literatura

Citovaná literatura

Knihy:

Čajda, O. *Objektové programování.* 1. vydání. Praha: Publishing a.s., 2009. ISBN: 978-80-247-2745-5

Merunka, V. *DATOVÉ MODELOVÁNÍ.* 1. vydání. Praha: Alfa Publishing, 2006. ISBN: 8086851540

Web:

Biek, M. *C# Preprocesor* , Stack Overflow, [online]. c2008, [cit. únor 2014]. dostupné z: http://stackoverflow.com/questions/37248/c-sharp-preprocessor,

Devbook.cz., Devbook.cz, [online]. [cit. září 2013]. dostupné z: http://www.devbook.cz/

ECMA. *ecma-international.org*., ecma-international.org., [online]. c2014 [cit. březen 2014]. dostupné z: http://www.ecma-international.org/default.htm

ECMA. *Standard ECMA-334*, ECMA.org., [online].c2006 [cit. březen 2014]. dostupné z: http://www.ecma-international.org/publications/standards/Ecma-334.htm

ECMA. *Standard ECMA-335*, ECMA.org., [online]. c2012 [cit. březen 2014] dostupné z: http://www.ecma-international.org/publications/standards/Ecma-335.htm

Gauld, A. *Co je programování?*, Jak se naučit programovat., [online]. [cit. září 2013]. dostupné: http://jaksenaucitprogramovat.py.cz/cztutwhat.html

Hordějčuk, i. V. *Objektové programování*, voho.cz., [online]. [cit. září 2013]. dostupné z: http://voho.cz/wiki/informatika/oop/

Ing. Petr Voborník, P. *Měnová kalkulačka*, ITnetwork.cz, [online]. [cit. červen 2014] dostupné z: http://www.itnetwork.cz/c-sharp-menova-kalkulacka-wpf-video-tutorial-webclient-download

Kučera, J. *historie programovacích jazyků*, Masarikova univerzita katedra informatiky, [online]. c2000 [cit. září 2013]. dostupné z: http://www.fi.muni.cz/usr/jkucera/pv109/2000/xkrubova.htm

Marek, B. *Programovací jazyk C#*, Katedra informatiky | FEI | VŠB-TU Ostrava, [online]. c2007 [cit. září 2013].dostupné z: http://www.cs.vsb.cz/behalek/vyuka/pcsharp/text/index.html

Microsoft. *http://www.visualstudio.com/*, Visual studio, [online]. c2014 [cit. únor 2014], dostupné z: http://www.visualstudio.com/cs-cz/explore/application-development-vs

Microsoft. *Introduction to the IDE (Visual C#)*, msdn.microsoft.com, [online]. c2008 [cit. březen 2014] dostupné z: http://msdn.microsoft.com/en-us/library/ms173064(v=vs.90).aspx

Microsoft. *msdn.microsoft.com*, Microsoft Developer network, [online]. c2013 [cit. březen 2014] dostupné z: http://msdn.microsoft.com/cs-cz#fbid=7NmjEqG_xvV

Microsoft. *Postupy: Seznámení s integrovaným vývojovým prostředím sady Visual Studio s použitím jazyka C# nebo Visual Basic*, msdn.microsoft.com, [online]. c2013 [cit. březen 2014] dostupné z: http://msdn.microsoft.com/cs-cz/library/jj153219.aspx

Parda, D. *STRUKTUROVANÉ PROGRAMOVÁNÍ*, isd.cz, [online]. c2009 [cit. říjen 2013] dostupné z: http://www.isd.cz/pascal/2strukt.html

Patnayak, S. *Preprocessor Directives in C#*, Code Project, [online]. c2012 [cit. únor 2014] dostupné z: http://www.codeproject.com/Articles/304175/Preprocessor-Directives-in-Csharp

Paul, R. *Introduction to Xamarin*, xamarin.com, [online]. c2013 [cit. duben 2014] dostupné z: http://xamarin.com/guide/

Puš, P. *Poznáváme C# a Microsoft.NET*, Zive.cz, [online]. c2004 [cit. březen 2014] dostupné z: http://www.zive.cz/clanky/poznavame-c-a-microsoft-net--1dil/sc-3-a-120978/default.aspx

TechieGeekx .Net Architecture., TechieGeekx the learner´s space, [online]. [cit. prosinec 2013], dostupné z: http://www.techiegeex.com/dotNet/dn001.php

wordpress. *NET Framework Fundamentals*, DevReminder, [online]. [cit. prosinec 2013] dostupné z: http:http://devreminder.wordpress.com/net/net-framework-fundamentals/

Xamarin. *http://mono-project.com.*, Mono-project, [online]. c2011 [cit. Leden 2014] dostupné z: http://www.mono-project.com/Compatibility

Xamarin. *Understanding the Xamarin Mobile Platform*, Xamarin.com, [online]. c2014 [cit. duben 2014] dostupné z, dostupné z: http://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_1_-_understanding_the_xamarin_mobile_platform/

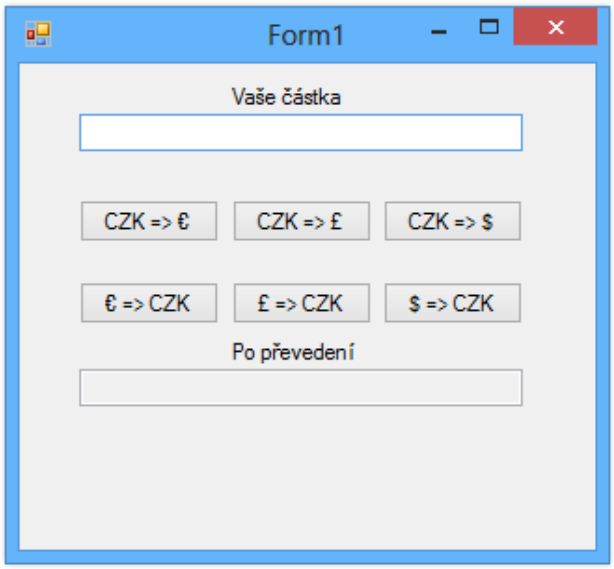
Xamarin. *Visual studio with xamarin*, Xamarin.com, [online]. c2014 [cit. duben 2014]dostupné z: http://developer.xamarin.com/guides/cross-platform/getting_started/visual_studio_with_xamarin/

Xamarin. *Xamarin Guide*, Xamarin.com, [online]. c2014 [cit. duben 2014]dostupné z: http://developer.xamarin.com/guides/

10 Přílohy

10.1 Grafický vzhled aplikace

10.1.1 Visual Studio



Obrázek 12: Výsledná forma aplikace pro MS Visual Studio (vytvořeno vlastním zpracováním)

10.1.2 Xamarin Studio

Obrázek 13: Výsledná forma aplikace pro Xamarin Studio (vytvořeno vlastním zpracováním)

10.2 Visual Studio zdrojový kód

10.2.1 Třída Form1.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Web;
using System.Net;

namespace Kurz_euro_CZK
{
    public partial class Form1 : Form
    {
        public Form1()
        {

            InitializeComponent();
            kurz ();
            Vstup.Text = "0";
        }
         string EuroKurz;
         string DolarKurz;
         string LibraKurz;
         char t;
         WebClient euro;
         WebClient dol;
         WebClient lib;




        private void CzknaEur_Click(object sender, EventArgs e)
        {

           string  a = Vstup.Text;
           string  b = EuroKurz;
            Vystup.Text = Prevod.knm(a, b);



        }

        private void EurnaCzk_Click(object sender, EventArgs e)
        {

            string a = Vstup.Text;
            string b = EuroKurz;
            Vystup.Text = Prevod.mnk(a, b);

        }

        private void CzknaDol_Click(object sender, EventArgs e)
        {
```

```
            string a = Vstup.Text;
            string b = Convert.ToString(DolarKurz);
            Vystup.Text = Prevod.knm(a, b);

        }

        private void DolnaCzk_Click(object sender, EventArgs e)
        {

            string a = Vstup.Text;
            string b = Convert.ToString(DolarKurz);
            Vystup.Text = Prevod.mnk(a, b);

        }
        private void CzknaLib_Click(object sender, EventArgs e)
        {


            string a = Vstup.Text;
            string b = Convert.ToString(LibraKurz);
            Vystup.Text = Prevod.knm(a, b);


        }

        private void LibnaCzk_Click(object sender, EventArgs e)
        {

            string a = Vstup.Text;
            string b = Convert.ToString(LibraKurz);
            Vystup.Text = Prevod.mnk(a, b);

        }
        private void kurz ()
        {

            euro = new WebClient();
             EuroKurz =
euro.DownloadString(String.Format("http://www.cnb.cz/miranda2/m2/cs/financni_trhy/devi
zovy_trh/kurzy_devizoveho_trhu/vybrane.txt?mena=EUR&od={0}&do={1}",DateTime.Today.AddD
ays(-6).ToString("dd.MM.yyyy"), DateTime.Today.ToString("dd.MM.yyyy")));
            EuroKurz = EuroKurz.Substring(EuroKurz.LastIndexOf('|') + 1);



              dol = new WebClient();
                DolarKurz =
dol.DownloadString(String.Format("http://www.cnb.cz/miranda2/m2/cs/financni_trhy/deviz
ovy_trh/kurzy_devizoveho_trhu/vybrane.txt?mena=USD&od={0}&do={1}",
DateTime.Today.AddDays(-6).ToString("dd.MM.yyyy"),
DateTime.Today.ToString("dd.MM.yyyy")));
                DolarKurz = DolarKurz.Substring(DolarKurz.LastIndexOf('|') + 1);


               lib = new WebClient();
               LibraKurz =
lib.DownloadString(String.Format("http://www.cnb.cz/miranda2/m2/cs/financni_trhy/deviz
ovy_trh/kurzy_devizoveho_trhu/vybrane.txt?mena=GBP&od={0}&do={1}",
DateTime.Today.AddDays(-6).ToString("dd.MM.yyyy"),
DateTime.Today.ToString("dd.MM.yyyy")));
               LibraKurz = LibraKurz.Substring(LibraKurz.LastIndexOf('|') + 1);
```

```
        }

        private void Vstup_KeyPress(object sender, KeyPressEventArgs e)
        {
                 t = e.KeyChar;
                if (!Char.IsDigit(t) && t != 8 && t != 46)
                {
                    e.Handled = true;
                }
        }

        private void Form1_Load(object sender, EventArgs e)
        {

        }



    }

}
```

10.2.2 Třída Převod.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Kurz_euro_CZK
{
    class Prevod
    {
        internal static string knm(string a, string b)
        {
            return Convert.ToString((Convert.ToDouble(a) * Convert.ToDouble(b)));
        }
        internal static string mnk(string a, string b)
        {
            return Convert.ToString((Convert.ToDouble(a) / Convert.ToDouble(b)));
        }

    }
}
```

10.3 Xamarin Studio zdrojový kód

10.3.1 Třída MainActivity.cs

```
using System;
using Android.App;
using Android.Content;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;
using System.Net;

namespace Meny
{
    [Activity (Label = "Meny", MainLauncher = true, Icon = "@drawabl
e/icon")]
    public class MainActivity : Activity
    {

        EditText vstup, vystup;
        Button EurCz, CzEur, LibCz, CzLib, DolCz, CzDol;
        string EuroKurz;
        string DolarKurz;
        string LibraKurz;
        WebClient euro;
        WebClient dol;
        WebClient lib;
        protected override void OnCreate (Bundle bundle)
        {
            base.OnCreate (bundle);

            kurz ();

            SetContentView (Resource.Layout.Main);
            vstup = FindViewById<EditText> (Resource.Id.editText1);
            vystup = FindViewById<EditText> (Resource.Id.editText2);

            EurCz = FindViewById<Button> (Resource.Id.button1);
            CzEur = FindViewById<Button> (Resource.Id.button2);
            LibCz = FindViewById<Button> (Resource.Id.button3);
            CzLib = FindViewById<Button> (Resource.Id.button4);
            DolCz = FindViewById<Button> (Resource.Id.button5);
            CzDol = FindViewById<Button> (Resource.Id.button6);

            EurCz.Click += eurnk;
            CzEur.Click += kneur;
            LibCz.Click += libnk;
            CzLib.Click += knlib;
            DolCz.Click += dolnk;
```

```
    CzDol.Click += kndol;
}

void eurnk (object sender, EventArgs e){

    string a = vstup.Text;
    string b = EuroKurz;
    vystup.Text = Prevod.mnk(a, b);



}

void kneur (object sender, EventArgs e){

    string a = vstup.Text;
    string b = EuroKurz;
    vystup.Text = Prevod.kmn(a, b);

}

void libnk (object sender, EventArgs e){

    string a = vstup.Text;
    string b = LibraKurz;
    vystup.Text = Prevod.mnk(a, b);



}

void knlib (object sender, EventArgs e){

    string a = vstup.Text;
    string b = LibraKurz;
    vystup.Text = Prevod.kmn(a, b);

}
void dolnk (object sender, EventArgs e){

    string a = vstup.Text;
    string b = DolarKurz;
    vystup.Text = Prevod.mnk(a, b);
}
void kndol (object sender, EventArgs e){

    string a = vstup.Text;
    string b = DolarKurz;
    vystup.Text = Prevod.kmn(a, b);
```

```
        }

        void kurz ()
        {

            euro = new WebClient();
            EuroKurz = euro.DownloadString(String.Format("http://www
.cnb.cz/miranda2/m2/en/financial_markets/foreign_exchange_market/exc
hange_rate_fixing/selected.txt?code=EUR&from={0}&to={1}",DateTime.To
day.AddDays(-
2).ToString("dd.MM.yyyy"), DateTime.Today.ToString("dd.MM.yyyy")));
            EuroKurz = EuroKurz.Substring(EuroKurz.LastIndexOf('|')
+ 1);


            dol = new WebClient();
            DolarKurz = dol.DownloadString(String.Format("http://www
.cnb.cz/miranda2/m2/en/financial_markets/foreign_exchange_market/exc
hange_rate_fixing/selected.txt?code=USD&from={0}&to={1}", DateTime.T
oday.AddDays(-
2).ToString("dd.MM.yyyy"), DateTime.Today.ToString("dd.MM.yyyy")));
            DolarKurz = DolarKurz.Substring(DolarKurz.LastIndexOf('|
') + 1);


            lib = new WebClient();
            LibraKurz = lib.DownloadString(String.Format("http://www
.cnb.cz/miranda2/m2/en/financial_markets/foreign_exchange_market/exc
hange_rate_fixing/selected.txt?code=GBP&from={0}&to={1}", DateTime.T
oday.AddDays(-
2).ToString("dd.MM.yyyy"), DateTime.Today.ToString("dd.MM.yyyy")));
            LibraKurz = LibraKurz.Substring(LibraKurz.LastIndexOf('|
') + 1);

        }

    }
}
```

10.3.2 Třída Převod.cs

```
using System;
using Android.App;
using Android.Content;
using Android.Runtime;
using Android.Views;
using Android.Widget;
using Android.OS;

namespace Meny
{
    public class Prevod
    {
        public Prevod ()
        {
        }
            internal static string mnk (string a, string b)
            {
                return Convert.ToString((Convert.ToDouble(a) / Conve
rt.ToDouble(b)));
            }
        internal static string kmn (string a, string b)
        {
            return Convert.ToString((Convert.ToDouble(a) * Convert.T
oDouble(b)));        }

        }
    }
```