THE INCLUSION OF DESIGN PATTERNS IN THE TEACHING OF PROGRAMMING

Rostislav FOJTÍK

Abstract: The aim of this paper is to present the possibilities of using design patterns in the teaching of programming. According to the performed analyzes the procedures and methodologies of teaching programming shows that Design Patterns are used only marginally. For these reasons, students learn to improper practices that subsequently applied in practical solutions programs. According to the experiments show that the correct use of the teaching of design patterns can improve student performance in programming.

Keywords: Design Patterns, object-oriented programming, teaching programming.

ZAŘAZENÍ NÁVRHOVÝCH VZORŮ DO VÝUKY PROGRAMOVÁNÍ

Resumé: Cílem příspěvku je představit možnosti využití návrhových vzorů ve výuce programování. Podle provedených rozborů postupů a metodik výuky programování je patrné, že návrhové vzory jsou používány spíše okrajově. Z těchto důvodu se studenti učí nevhodným postupům, které následně uplatňují při praktickém řešení programů. Z provedených experimentů vyplývá, že správné využití výuky návrhových vzorů může zlepšit schopnosti studentů v oblasti programování.

Klíčová slova: návrhové vzory, objektově orientované programování, výuka programování

1 Úvod

Aktuálně využívané metodické postupy ve výuce programování se značně liší. Vyučující často vycházejí z metodiky, kterou sami byli vyučováni, ale která není nejvhodnější pro nová programovací paradigmaty. V současné době se můžeme setkat například s následujícími paradigmaty v oblasti programování:

- strukturované,
- imperativní,
- funkcionální,
- logické,
- objektově orientované.

S funkcionálním a logickým paradigmatem v oblasti výuky programování se nesetkáme téměř na žádné ze základních a středních škol. Základní školy se obvykle věnují pouze výuce základům algoritmizace. Výuka programování se objevuje spíše v náplni volitelných a zájmových kroužků. Na středních a vyšších odborných školách lze v učebních plánech najít předměty přímo zaměřené na programování a mnohdy s dostatečně rozsáhlou hodinovou dotací. Počet vyučovacích hodin přidělených však mnoho nevypovídá o metodice a kvalitě výuky programování. Proto v tomto příspěvku jsou předloženy rozborů některých tematických plánů, zkušenosti získané z praktické výuky, pedagogických experimentů a dříve provedených průzkumů.

2 Analýza tematických plánů

Průzkumem na 63 středních a 71 základních školách v Moravskoslezském kraji bylo zjištěno, že 84 % středních škol a 32 % základních škol se věnuje výuce programování [3] [4]. Kromě toho byl proveden rozbor školních vzdělávacích programů (ŠVP), učebních plánů a tematických plánů vybraných 27 středních škol. Z toho bylo 14 gymnázií a 13 středních průmyslových nebo odborných škol. Na průmyslových a odborných školách je výuka programování velmi ovlivněná konkrétními studijními obory. U některých z nich je výuka zaměřena pouze na základy algoritmizace. Část oborů vyučuje konkrétní programovací jazyky, které jsou nutné například pro programování CNC obráběcích strojů nebo mikročipů v elektronických zařízeních. Učební plány studijních oborů zaměřených na informatiku obsahují předměty programování, které se obvykle vyučují 2 až 3 roky. Podobně i část gymnázií má ve svých učebních plánech předměty, ve kterých je nezanedbatelná část hodin věnována algoritmizaci a programování. Na gymnáziích se programování obvykle vyučuje ve volitelných seminářích.

Následující graf ukazuje počty škol vyučujících programování nebo algoritmizaci jako samostatný předmět nebo v rámci jiných předmětů.

Obr. 1: Zastoupení výuky programování ve výuce na sledovaných SŠ

Podíváme-li se podrobněji na obsahy tematických plánů, zjistíme, že objektově orientované programování je součástí tematických plánů pouze v sedmi ze sledovaných středních škol. A to i přes skutečnost, že na 17 sledovaných školách mají samostatný předmět zaměřený přímo na programování.

Skladba tematických plánů není často optimální a nepodporuje vhodnou metodiku výuky. Obvyklé chyby jsou popsány v následně uvedených ukázkách.

Ukázka č. 1 – gymnázium

Výuka programování v jazyku Pascal v rozsahu 19 hodin. Což při dotaci 2 hodin týdně odpovídá asi

2,5 měsícům výuky. Za tak krátkou dobu se žáci většinou nestačí naučit programovací jazyk důkladně a seznámí se pouze se základními konstrukcemi jazyka bez schopnosti je smysluplně využít. Z praktických zkušeností vyplývá, že žáci středních škol potřebují pro dobré pochopení a naučení konkrétního programovacího jazyka alespoň jeden školní rok. Teprve pak jsou schopni programovací jazyk správně používat a tvořit rozsáhlejší programy. Při malé hodinové dotaci, se žáci příliš soustředí na zvládnutí jazyka a mnohdy nepochopí podstatné principy tvorby algoritmů a programů. Proto se jeví jako vhodnější při malé časové dotaci využívat i na středních školách mikrosvětly (Logo, Karel, Scratch) nebo programovat robotické stavebnice (Lego). Programování je pomocí těchto nástrojů pro žáky mnohem přitažlivější a názornější. Lépe pochopí, k čemu jednotlivé algoritmické konstrukce prakticky slouží a jak je správně využít. Programovací jazyk (stejně jako jakýkoliv jiný jazyk) se obvykle nedá zvládnout za krátkou dobu, ale vyžaduje to delší čas. Žáci se musí kromě pochopení logických souvislostí naučit mnoha dovednostem, a to lze většinou jen dostatečně častým opakováním.

Nevhodně jsou často zakomponované do výuky objektově orientované vlastnosti programování. Tvůrci je obvykle zařazují až jako

poslední kapitoly nebo je tématu věnováno jen velmi málo času. Principy objektově orientovaného programování jsou mnohými tvůrci tematických plánů chápány jen jako další funkcionalita a rozšíření programovacího jazyka. Nevnímají, že se jedná o zcela jiné paradigma, které vyžaduje naprosto jiné postupy tvorby programů. Mnoho vyučujících rovněž chybně ztotožňuje objektově orientované programování pouze s návrhem uživatelského rozhraní aplikace a využitím grafických komponent.

Ukázka č. 2 – střední průmyslová škola

Typickým příkladem je jeden ze zkoumaných tematických plánů ze střední průmyslové školy strojní a elektronické, která nabízí žákům volitelný předmět s názvem Programování. Hodinová dotace je 1 hodina přednášek a 3 hodiny cvičení týdně. Hned první kapitola je sice věnována základním principům objektově orientovaného programování, ale jedná se jen o krátký úvod do problematiky. Většina témat je věnována pouze zvládnutí práce s vizuálním vývojovým prostředím a využitím grafických komponent pro uživatelské rozhraní aplikace. V plánu chybí jakákoliv zmínka o architektuře aplikací, chybí podrobnější rozbor objektově orientovaných vlastností (dědičnost, kompozice, polymorfismus, interface...).

Žádný z analyzovaných tematických plánů neobsahuje témata zabývající se výukou a praktickým využitím návrhových vzorů, architekturou programů, jejich analýzou a návrhem.

3 Zaměření výuky programování

Pomocí nestrukturovaného interview s 23 učiteli informatiky, kteří na katedře informatiky a počítačů Ostravské univerzity studovali rozšiřující studium informatiky, bylo zjištěno, že ve výuce programování existují dva extrémní přístupy:

První typ vyučujícího se zaměřuje hlavně na návrh uživatelského rozhraní a snaží své žáky naučit co nejvíce grafických komponent. Učitel dává přednost vizuálním vývojovým nástrojům, neboť pro žáky je tvorba programů zábavnější a jsou více motivováni.

Opačný extrém představuje učitel, který požaduje po žácích vytvářet programy pouze pro konzolu, tedy s výstupem do příkazového řádku. Programy většinou řeší více či méně složité matematické úkoly. Většině žáků chybí motivace, výuka programování je příliš nebaví a moc tématu nerozumí.

Mezi vyučujícími chybí skupina, která by se důkladněji zaměřovala na analýzu, návrh a architekturu programu. Z těchto důvodů, vytvářejí žáci často nevhodně koncipované programy. Nejvíce je to znatelné tam, kde využívají k výuce

programování vizuální vývojové nástroje typu Delphi a podobné. Žáci se podrobně učí, jak používat konkrétní grafické komponenty, ale již méně času se věnuje správné analýze zadání, objektovému návrhu, architektuře programu a vhodné implementaci [6].

Je tedy vůbec vhodné používat vývojová prostředí s nástroji pro vizuální návrh uživatelského rozhraní ve výuce?

Využívání vizuálních vývojových prostředí ve výuce programování na středních školách sebou nese některá nebezpečí a nevýhody:

- metodicky nedostatečně vzdaný učitel často sklouzne pouze k výuce tvorby grafického uživatelského prostředí aplikace;
- žáci po pár hodinách získají pocit, že programování je jednoduché a stačí jen „klikat“ myší a aplikace je hotova;
- některá vývojová prostředí nedostatečně podporují tvorbu vhodné architektury programů;
- pro některé žáky jsou vizuální vývojové nástroje složité a nepřehledné.

Naopak mezi silné stránky a výhody vizuálních vývojových nástrojů ve výuce programování patří:

- žáci programují „opravdové“ aplikace (programy lze reálně využívat), což vede k jejich větší motivovanosti;
- žáci se naučí pracovat s moderními vývojovými nástroji;

- žáci mohou používat a pochopit moderní programovací postupy.

Využití vizuálních nástrojů pro tvorbu aplikací je ve výuce určitě vhodné, jen se nesmí stát cílem, ale pouhým prostředkem a nástrojem. Tematické plány a cíle výuky programování by měly být směřovány hlavně ke tvorbě správné architektury vytvářených aplikací. K tomu může velkou měrou přispět větší podpora výuky návrhových vzorů.

4 Podpora výuky architektury programů

Jak je důležité zařadit do výuky programování témata věnující se správné analýze a návrhu programů je vidět například na výsledcích všech ročníků soutěžní přehlídky programů pro studenty gymnázií z Moravskoslezského kraje [8]. Soutěžící patří na svých školách mezi nejlepší studenty v předmětu informatika (programování) a svým programům věnují spoustu času a energie. Studenti obvykle své soutěžní práce tvoří samostatně mimo výuku. Mnohdy je možné se setkat se zajímavými nápady, které svědčí o invenci jejich tvůrců. Bohužel na mnohých provedeních je vidět, že se studenti během svého studia nesetkali s informacemi o tom, jak by měl vypadat návrh architektury programů, nedozvěděli se nic o návrhových vzorech a analýze programů. Obvykle své programy řeší systémem pokus-omyš

a velká část programů přihlášených do soutěže má nevhodnou strukturu. Je samozřejmé, že se od programů nečeká profesionální úroveň, ale není žádoucí, aby studenti získávali špatné návyky.

Zvláště je to vidět na studentech, kteří použili k vývoji vizuální vývojový nástroj Delphi a podobné. Prezentační, řídicí i datovou logiku obvykle míchají v jedné unitě (modulu). Přitom stačí, když se studenti během výuky seznámí alespoň s některými základními architektonickými a návrhovými vzory. Vytváří-li studenti programy pro grafické operační systémy, měli by se minimálně dozvědět o architektuře Model-View-Controller (MVC), která rozděluje datovou část od uživatelského rozhraní a řídicí logiky. Mnohé z frameworků jsou navrženy a implementovány tak, aby architekturu MVC co nejlépe podpořovaly (například Cocoa pro Mac OS X a iOS).

Obr. 2: Příklad architektury MVC

Důsledné vyžadování architektury MVC vede studenty k dobrým návykům a jejich aplikace častěji odpovídají správné struktuře. Z rozhovoru s vítěznými studenty se prokázalo, že ti, kteří měli možnost se seznámit s návrhovými vzory, se lépe orientovali v návrhu architektury aplikace a věnovali ji velkou část energie při řešení.

5 Návrhové vzory

Návrhové vzory (Design Patterns) jsou doporučené postupy k řešení úloh. Poskytují programátorům mnoho výhod, jako je například podpora opětovného využití návrhu. Existuje základních 23 návrhových vzorů, které se dají rozčlenit do tří skupin [7]:

- vytvářecí vzory (Creational Patterns)
- strukturální vzory (Structural Patterns)
- vzory chování (Behavioral Patterns)

Návrhové vzory jsou v profesionální programátorské praxi relativně dobře známy a využívány. Kdy je ovšem zařadit do výuky? Podíváme-li se do běžných učebnic programování, zjistíme, že téma návrhových vzorů je často opomíjeno vůbec nebo je zařazeno až na úplný závěr. Podobně jsou koncipovány výukové kurzy programování. Nejprve se vyučující věnují algoritmizaci, základům programovacího jazyka a teprve na konci se účastníci výuky dozví o principech

objektově orientovaného programování a občas několik málo informací o návrhových vzorech.

Druhý přístup k výuce vyčlení návrhové vzory jako samostatný celek a celý výukový kurz se věnuje pouze návrhovým vzorům a jejich využití.

V poslední době se začíná při výuce programování uplatňovat třetí přístup, který již od počátku dává důraz na architekturu programu a tedy i vhodné používání návrhových vzorů [5]. Při tom se studenti paralelně dozvídají jednotlivé prvky programovací jazyka a algoritmické postupy. Absolventi takto pojatých kurzů obvykle lépe chápou principy správného návrhu programu.

O důležitosti správného návrhu programů svědčí mnoho příkladů. Následující příklad se objevuje často v řešení studentů, kteří se málo setkali s výukou návrhu architektury programů a využitím vzorů. Příklad ukazuje, jak nesprávně navržené třídy, jejich vlastnosti, schopnosti a vztahy mezi nimi, značně komplikují funkčnost aplikace. Představme si, že studenti mají navrhnout třídy a objekty, které budou popisovat členy softwarového týmu. Můžeme zde identifikovat analytiku, programátory, testery, manažery atd. Všechny tyto třídy mají společné vlastnosti a schopnosti, proto i začátečníka napadne navrhnout společného rodiče. Studenti po seznámení

s dědičností obvykle navrhnou následující řešení:

Obr. 3: Diagram tříd nesprávného návrhu

Celý návrh bude vypadat zdánlivě správně. Vytvoříme instanci třídy *Programmer*, která bude představovat jednoho konkrétního programátora týmu.

```

Programmer prog;
prog = new Programmer();
  
```

Problém nastane v okamžiku, kdy programátor změní pracovní zařazení a stane se například manažerem. V té chvíli bude potřeba vytvořit novou instanci, a to pro třídu *Manager*. Překopírovat do ní všechny vlastnosti z objektu *prog* a následně tento objekt zrušit. Tento postup však zásadně neodpovídá reálnému procesu v požadovaném informačním systému. Jako výhodnější se jeví využít místo dědičnosti agregace a uvědomit si, že zaměstnanec má určitou roli v zaměstnání. Při změně zařazení zaměstnance mu změním jeho roli a není potřeba vytvářet nové objekty [1].

Obr. 4: Diagram tříd řešení aplikace

6 Příklady využití návrhových vzorů

Na dalším příkladu, ve kterém simulujeme obchod s tropickým ovocem, můžeme ukázat výhody využití návrhových vzorů ve výuce. Představme si, že je potřeba vytvořit program, který bude řídit nakupování tropického ovoce v konkrétní zemi, a to vždy podle ročního období. Pokud se žáci seznámí s objektově orientovaným programováním, obvykle je napadne navrhnout třídy ProductA (například ovoce z Jižní Afriky) a ProductB (ovoce ze Španělska). Pokud jejich povědomí o objektově orientovaném návrhu bude větší než jen znalost pojmů objekt a třída, navrhnou pro tyto třídy společného předka nebo rozhraní. V hlavním programu se pak objeví následné řešení, kdy v závislosti na konkrétním měsíci v roce se budou vytvářet různé objekty (nakupovat ovoce v různých zemích):

```

for(int month=1; month<12; month++)
{

```

```

    if (month >=5 && month
    <=9)
    {
        ProductB p = new Pro-
        ductB();
        p.fromCountry();
    }
    else
    {
        ProductA p = new Pro-
        ductA();
        p.fromCountry ();
    }
}

```

Řešení vypadá v pořádku jen do té doby, než budeme potřebovat program upravit pro další podmínky nákupu tropického ovoce. Například nákup z jiných zemí a v závislosti na dalších vnějších okolnostech. V předcházejícím kódu bude potřeba připsat další podmínky a znatelně zasahovat do hlavního algoritmu.

Jako vhodnější se jeví využití návrhového vzoru *Factory Method* (Tovární metoda) [2]. UML diagram tříd ukazuje následující obrázek.

Obr. 5: Diagram tříd vzoru Tovární metoda

Součástí vzoru je třída Creator, jejíž objekt může v našem programu sloužit jako nákupčí tropického ovoce. Odpovědnost o správný nákup v konkrétním měsíci přesuneme z hlavního programu na tento objekt. Jakákoliv změna podmínek je záležitostí pouze úpravy třídy Creator, hlavní program zůstává nezměněn.

```
Creator buyer = new Creator();
for(int month=1; month<=12; month++)
{
    prod =
    buyer.FactoryMethod(month);
}
```

Z praktických experimentů ve výuce, které byly prováděny ve výuce programování na katedře informatiky a počítačů Ostravské univerzity v předmětech Programování v C++, Vývoj mobilních aplikací pro iOS a v kurzech programování pro středoškolské studenty, bylo zjištěno, že studenti, kteří se v kurzu od počátku setkávají s problematikou architektury programů a s návrhovými vzory, mají menší problémy se správným návrhem svých programů. Návrhové vzory nechápou jen jako jakousi nadstavbu programování, ale jako jeho důležitou a neoddělitelnou součást. Jako příklad může sloužit kurz pro žáky středních škol.

Účastníci kurzu byli z různých středních škol. Z hlediska znalosti programování se jednalo o nesourodou skupinu. 4 žáci neměli žádné

znalosti z programování, 11 účastníků se učilo strukturovanému programování a 4 žáci měli základní znalosti v oblasti objektově orientovaného programování. Nikdo z účastníků neměl zkušenosti s vývojem pro iOS ani s prací na počítačích s operačním systémem Mac OS X. Cílem kurzu bylo ukázat žákům postupy, které budou založené na vhodné architektuře, návrhových vzorech a využití frameworků. Tedy na postupech, které jsou v současné době uplatňovány při vývoji aplikací v profesionální praxi. Kurz byl sestaven z pěti setkání, které trvaly tři hodiny.

1. seminář

Seznámení s iOS a Mac OS X. Třída, objekt, zapouzdření, atributy, zprávy. Využití frameworků. Cocoa Touch. Základní seznámení s programovacím jazykem Objective-C.

2. seminář

Xcode – prostředí pro vývoj. První jednoduchá aplikace – převodník jednotek rychlosti. Model – View – Controller. Dědičnost. Delegation. Návrhový vzor Command Pattern – Target-Action. Návrhový vzor Template Method – Two-Stage Creation. Accessors – getter, setter.

3. seminář

Ukázková aplikace – prohlížeč obrázků. Komponenty: Picker View, Table View. Protocols. Návrhový vzor Strategy Pattern – Delegates, Data Sourcing.

4. seminář

Ukázková aplikace – využití lokace a map. Framework MapKit. Návrhový vzor Singleton Pattern – NSApplication class.

5. seminář

Ukázková aplikace – ukládání dat do souboru a iCloudu. Databáze SQLite. Core Data.

Vzhledem k malé časové dotaci, nebylo možné v rámci výuky v kurzu důkladně probrat všechny architektonické a návrhové vzory. Účastníci rovněž zvládli pouze základy programovacího jazyka Objective-C. Pro mnohé studenty byly první lekce dosti obtížné. Největší problémy měli účastníci, kteří se zatím s programováním nesetkali. Tito žáci by potřebovali více času pro svou práci a více lekcí, aby zvládli hlavně dovednost práce ve vývojovém prostředí. Studenti, kteří absolvovali všechny semináře, chápali nutnost příhodné architektury programu (MVC), výhody využití návrhových vzorů a frameworků a při vytváření vlastních aplikací se snažili dodržovat vhodné postupy.

7 Shrnutí

Ukazuje se, že využití návrhových vzorů ve výuce programování je vhodné nejen na vysokých školách, ale rovněž i na nižších stupních. A to v případě, že cílem výuky není jen pouhá algoritmizace, ale komplexní schopnost programování.

Návrhové vzory představují v současné době jeden z mocných nástrojů při návrhu a následné implementaci programů. Z hlediska výuky by neměly být zařazovány až jako poslední kapitoly učebnic a výukových kurzů, ale je vhodnější, aby prolínaly výkladem již od počátku. Aplikace návrhových vzorů ve výuce umožní nejen žákům, ale rovněž vyučujícím lépe pochopit a definovat správně cíle výuky programování.

8 Literatura

- [1] ARLOW J., NEUSTAD I. *UML a unifikovaný proces vývoje aplikací*, Computer Press, Brno 2003, ISBN 80-7226-947-X
- [2] BISHOPOVÁ J. *C# návrhové vzory*, Zoner Press, Brno 2010, ISBN 978-80-7413-076-2
- [3] FOJTÍK, R., JIRÁSKOVÁ, D. Programování na základních a středních školách. *Objekty 2009*. Gaudeamus, Univerzita Hradec Králové, 2009. s. 75–83. ISBN 978-80-7435-009-2.
- [4] FOJTÍK, R., DROZDOVÁ, M. Teaching of programming at secondary schools. *ICTE 2009*. Ostrava: University of Ostrava, 2009. s. 77–81. ISBN 978-80-7368-459-4.
- [5] GOVENDER I., From Procedural to Object-Oriented Programming (OOP) – An exploratory study of teachers' performance, *SACJ*, vol. 46. 2010. ISSN 1015-7999.
- [6] SAELI M., PERRENET J.,

JOCHEMS J., ZWANEVELD B.,
Teaching Programming in Secondary School: A Pedagogical Content Knowledge Perspective, *Informat-ics in Education – An International Journal*, vol. 10_1/2011, ISSN 1648-5831

[7] STERKIN A., Teaching Design Patterns, (online), URL:
<<http://ww1.ucmss.com/books/LFS/CSREA2006/FEC4388.pdf>>

[8] Soutěžní přehlídka studentských program, (online), URL:
http://www.gvoz.cz/akce/2013_sps_p.php

Mgr. Rostislav Fojtík, PhD.
Katedra informatiky a počítačů
Přírodovědecká fakulta UO
30. dubna 22
701 00, Ostrava, ČR
Tel: +420 603 167 768
E-mail: rostislav.fojtik@osu.cz
www pracoviště:
<http://prf.osu.cz/kip/>