

# Výuka programování pro praxi

Rudolf Pecinovský

ICZ a.s., 104 00 Praha 4, Hvězdova 1689/2a  
VŠE, Fakulta informačních technologií, 130 67, Praha 3, nám W. Cuhurchilla 4  
rudolf@pecinovsky.cz

## 1 Úvod

Procházíme-li učební plány škol vychovávajících budoucí vývojáře, analytiku, testery a další profese zabývající se vývojem softwaru, najdeme v nich řadu předmětů seznamujících studenty s nejrůznějšími speciálními oblastmi programování a probírajících klíčové algoritmy a architektonické principy používané v dané oblasti. Populární jsou i předměty seznamující studenty s některými nejnovějšími technologiemi. Málokdy se však v těchto učebních plánech nesetkáme s předměty, v nichž by se studenti mohli seznámit s nejnovějšími obecnými zásadami programování nebo se zásadami, jejichž dodržování zvýší prodejnost vyvíjeného produktu.

## 2 Studijní plány

Podíváte-li se na styl výuky programování na informaticky zaměřených fakultách, které samy sebe prohlašují za prestižní, zjistíte, že na řadě z nich vyvolává styl výuky v prvním semestru intenzivní představu, že termín *objekt* je zde považován za neslušné slovo.

V druhém semestru se sice studenti dozvědí, že něco takového existuje, ale vyučující se soustředí především na výklad nejrůznějších syntaktických konstrukcí, jejichž použití je často demonstrováno na příkladech, které by v učebnicích skutečného objektově orientovaného programování mohly úspěšně vystupovat v roli odstrašujících příkladů, jak se objektově programovat nemá.

Absolventi těchto kurzů odcházejí s představou, že programovat objektově znamená používat v programu třídy a objekty, že nejlepší objektově orientovaná konstrukce je dědičnost a že rozhraní (pokud se o něm přednášející vůbec stihne zmínit) je konstrukce, která přináší chudou náhražku násobné dědičnosti do jazyků, jejichž autoři si násobnou dědičnost netroufali implementovat.

V dalších semestrech se sice občas objeví předměty, které by mohly uvést vše na pravou míru (např. předměty seznamující s návrhovými vzory), avšak na rozdíl od vstupních

kurzů již tyto předměty zasáhnou jenom část studentů. Navíc publikované materiály vyvolávají ve čtenářích představu suchého katalogu majícího za cíl studenty pouze seznámit s tím, že něco takového existuje, když už se o tom ve světě všude mluví. Většinou však nevyvolávají dojem, že by vyučující chtěl naučit studenty opravdu „myslet objektivně“ a ukázat jim, že návrhové vzory a celá moderní koncepte programování přicházejí s poněkud jiným paradigmatem, než je to, které jim bylo předváděno v úvodních kurzech programování v minulých semestrech. Můžeme jen doufat, že se jedná opravdu pouze o dojem ze zveřejněných materiálů a vlastní výuka probíhá v jiném duchu.

### **3 Výuka zapomíná na zákazníka**

Nepříjemným vedlejším efektem výuky, která se soustředí na technologie a ignoruje změnu celkového klimatu ve vývoji softwaru, je to, že student odchází ze školy s pocitem, že ze všeho nejdůležitější je vytvářet geniální programy. Zákazník je pro něj někdo, kdo neustále obtěžuje s dalšími požadavky. Nikdo mu totiž nějak nestačil říct, že tím, kdo jej živí, není ve skutečnosti jeho zaměstnavatel, ale zákazník, pro nějž je software vyvíjen.

Současný rozmach informačních technologií a vznik firem specializujících se na vývoj aplikací nejrůznějšího druhu poněkud pozměnil role. Nyní již zákazníci, kteří by si rádi objednali vývoj nějaké aplikace, nehledají firmu, která by jim byla ochotna jejich aplikaci naprogramovat, ale naopak softwarové firmy se ucházejí o zákazníky, kteří by byli ochotni si u nich vývoj aplikací objednat.

Chceme-li zákazníka přesvědčit, aby si objednal vývoj programu právě u nás, musíme mu nabídnout něco, co mu konkurence nenabízí. Řada programátorů se domnívá, že tím něčím je dokonalá aplikace. V současné době se však již rozdíl v „dokonalosti“ jednotlivých řešení dostávají za mentální obzor běžného zákazníka, takže se podle nich těžko může rozhodnout. Firmy mu proto musí nabídnout něco, čemu zákazník rozumí a co jej osloví. Takovou veličinou může být např. magická zkratka TCO (Total Cost of Ownership – celková cena vlastnictví) charakterizující veškeré náklady spojené s pořízením, vlastnictvím, provozem a údržbou daného programu.

Dalším faktem, který může příznivě ovlivnit rozhodnutí zákazníka o přidělení zakázky, je jeho předchozí zkušenost s danou či konkurenční firmou a její schopností pružně reagovat na jeho průběžně se měnící požadavky. Budou-li naši programátoři reagovat na modifikační požadavky standardní odpovědí typu: „Tak, jak jsem to naprogramoval, je to nejlepší. Ať

s tím zkusí chvíli pracovat a pozná to sám. Nebudu svůj program prznit.“, těžko zákazníka přesvědčíme, aby si u nás objednal i příští zakázku.

Na schůzkách manažerů softwarových firem jsem již několikrát slyšel o skvělých týmech absolventů MFF, které musela firma rozpustit, protože jí jejich „geniální programy“ nebyly schopny uživit. Najmuli si proto absolventy méně prestižních fakult, jejichž programy možná nebyly tak sofistikované, ale dokázali je udělat právě takové, jaké zákazník požadoval.

Nezasvěcenému by se možná mohlo zdát, že jsem se v této kapitole zcela odpoutal od tématu probíraného v kapitole předchozí. Opak je však pravdou. Vznik a vývoj řady moderních programovacích technik a metodik byl totiž vyvolán právě snahou co nejlépe vyjít vstříc zákazníkovi a získat tak konkurenční výhodu.

## 4 Co učit

Vraťme se však k původnímu problému a podívejme se znovu na otázku *Co učit?* Otázku bychom mohli formulovat i jinak: chceme naše studenty naučit skvěle programovat a vyhrávat programátorské soutěže, anebo je chceme naučit programovat tak, aby se o ně přetahovali zákazníci a aby si proto mohli programováním skvěle vydělávat?

Touží-li naši studenti především po tom, aby se naučili vyvíjet skvělé programy na které budou hrdí a získají tak věhlas mezi „stejně postiženými“ kolegy, je vhodné se soustředit na výklad různých specializovaných partií, v nichž studenty seznámíme s algoritmy a technikami, které zůstanou ostatním utajeny.

Jsou-li naši studenti poněkud materialističtější zaměření a neláká-li je pouze programování jako intelektuální výzva, ale láká-li je také jako potenciálně slibný zdroj příjmů, a to i za cenu toho, že se budou muset přizpůsobit požadavkům zákazníků, měli bychom je učit i tomu, jak zařídit, aby se jimi vytvářené programy dobře prodávaly.

Nechápejte předchozí odstavec tak, že bychom měli učit programátory prodávat software – tuto činnost bychom měli přenechat obchodníkům, protože vyžaduje přece jenom jiné spektrum schopností a dovedností. My bychom je měli pouze naučit vyvíjet programy tak, aby se pak obchodníkům dobře prodávaly. Měli bychom je proto učit vytvářet programy, které je možno snadno a rychle přizpůsobovat neustále měnícím se požadavkům zákazníka a které lze snadno a levně spravovat.

Nemohu tvrdit, že by programátoři, kteří se soustředí především na dokonalost kódu, nebyli dobře placeni (pokud jsou zrovna zaměstnání). Jejich častou nevýhodou však bývá

oslnění vlastní dokonalostí. Když jim zákazník vrátí jejich skvělý program, protože jej chce poněkud upravit a přizpůsobit svým novým požadavkům, nezdědka prohlásí, že zákazník je hlupák, protože když bude jejich program chvíli používat, sám pozná, že nabídnuté řešení je to nejlepší.

Takovou odpověď ale zákazník nechce slyšet. Nezdědka se tak stává, že firma nakonec musí svůj tým vynikajících, avšak nepřizpůsobivých programátorů rozpustit, protože tito programátoři jsou pro ni ztrátoví. Původní tým proto nahradí týmem programátorů, kteří jsou možná méně skvělí, avšak kteří firmu (a tým i sebe) skvěle užijí, protože jsou schopni reagovat na rozmary zákazníka (nebo je dokonce předvídat) lépe než kdokoliv jiný.

## 5 Jak učit a neučit

Zde se znovu dostáváme k moderním technikám a metodikám programování a jejich výuce. Víme-li, že tyto techniky a metodiky byly opravdu vyvíjeny s cílem získat konkurenční výhodu, měli bychom našim studentům umožnit, aby se s nimi nejenom seznámili, ale aby je doopravdy přijali za své. Aby se jim, lidově řečeno, dostaly pod kůži.

Mají-li se studenti ztotožnit s přednášenými metodikami, není vhodné, aby se s těmito metodikami začali seznamovat až někdy před koncem studia a v předchozích semestrech pracovali podle metodik, které jsou s nimi tu v menším, tu ve větším rozporu. Musíme je proto od samého začátku učit pracovat a přemýšlet tak, aby se v pozdějších etapách výuku nemuse-li přeškolovat na nějaký jiný způsob práce.

Drtivá většina „dobře placených“ projektů je programována objektově. Mají-li být naši studenti schopni kvalifikované účasti na takovýchto projektech, měli bychom je od samého začátku výuky učit programovat objektově. Znovu zdůrazňuji, že programovat objektově neznamená pouze používat v programu třídy a objekty, ale skutečně při vývoji programu přemýšlet podle současných zásad OOP.

Zkušenost ukazuje, že jedinou současnou metodikou, která vyhovuje předchozím požadavkům, je metodika *Design Patterns First* vysvětlovaná např. v [3].

## 6 Závěr

Príspevek ukazuje některé rozpory mezi tím, co se v kurzech programování na našich univerzitách učí, a tím, jaké absolventy praxe vyžaduje. Upozorňuje na to, že skvělý program ještě nemusí firmu skvěle „uživit“. Připomíná, že zákazníci se nerozhodují především podle

kvality programu, ale upřednostňují jiná kritéria. Současně ukazuje, že moderní metodiky programování jsou vyvinuty právě s ohledem na tyto požadavky a že bychom je proto měly začleňovat do výuky časněji a intenzivněji, než je v současné době zvykem. V závěru konstatuje, že jedinou metodikou výuky, která těmto požadavkům vyhovuje, je metodika *Design Patterns First*.

## 7 Reference

- [1] PAVLÍČKOVÁ Jarmila, PAVLÍČEK Luboš: Zkušenosti s přístupem object-first v úvodním kurzu programování. *Objekty 2005 – sborník příspěvků devátého ročníku konference*, VŠB Ostrava, 2005. ISBN 80-248-0595-2.
- [2] PECINOVSKÝ Rudolf: *Návrhové vzory*, CPress, 2007. ISBN 978-80-251-1582-4.
- [3] PECINOVSKÝ, Rudolf: Aplikace metodiky Design Patterns First. *Objekty 2006 – sborník příspěvků desátého ročníku konference*, ČZU, Praha 2006. ISBN 80-213-1568-7.
- [4] PECINOVSKÝ Rudolf: Výuka programování podle metodiky Design Patterns First. *Tvorba softwaru 2006 – sborník přednášek*. ISBN 80-248-1082-4.
- [5] PECINOVSKÝ Rudolf, PAVLÍČKOVÁ Jarmila, PAVLÍČEK Luboš: Let's Modify the Objects First Approach into Design Patterns First, *Proceedings of the Eleventh Annual Conference on Innovation and Technology in Computer Science Education*, University of Bologna 2006.
- [6] PECINOVSKÝ, Rudolf: Začlenění návrhových vzorů do výuky programování. *Objekty 2005 – sborník příspěvků devátého ročníku konference*, VŠB Ostrava, 2005. ISBN 80-248-0595-2.
- [7] PECINOVSKÝ Rudolf: *Myslíme objektově v jazyku Java 5.0*, Grada, 2004. ISBN 80-247-0941-4.

**Abstrakt.** Již delší dobu však softwarové firmy prohlašují, že jim současný typický profil absolventa programátora nevyhovuje a potřebovali by absolventy s poněkud jiným profilem, než jaký jim školy nabízejí. Příspěvek se zabývá základními disproporcemi mezi tím, co školy nabízejí a co praxe vyžaduje.

Většina školních a soukromých úvodních kurzů programování se soustředí především na výklad základních vlastností jazyka a nejdůležitějších knihoven doplněný případně výkladem základů algoritmizace.

Naprostá většina školních i placených kurzů programování se soustředí na to, jak studenty naučit vymýšlet co nejlepší algoritmus, ty pokročilejší pak učí, jak vymyslet co nejchytřejší architekturu. Při tomto výkladu vyučující často zapomínají na to, že většina jejich studentů se chce programováním živit. V jejich kurzech proto chybí pasáže, které studentům ukazují, jak je třeba programovat, aby vytvářené programy byly co nejlépe prodejné. Příspěvek se zabývá některými z těchto aspektů, ukazuje, na které součásti programování by se nemělo zapomínat.

**Klíčová slova:** OOP, návrhové vzory, výuka programování, vstupní kurzy programování, design patterns first

## Annotation

The most university courses of programming concentrates on teaching students how to make up excellent algorithms or clever architecture. By this explanation the teachers often forget

that most of their students want to live on programming. Their courses therefore miss the subjects demonstrating students how to create programs that are also excellently saleable. The article shows some consequences of such kind of teaching and reminds which parts of programming science shouldn't be omitted in our teaching.