VÝUKA INFORMAČNÍCH TECHNOLOGIÍ NA VYSOKÝCH ŠKOLÁCH

Miroslav Virius
Fakulta jaderná a fyzikálně inženýrská, České vysoké učení technické v Praze
miroslav.virius@fjfi.cvut.cz

ABSTRAKT:
Příspěvek se zabývá problematikou výuky informačních technologií na vysokých školách technických, zejména se zřetelem k výuce programování. Rozebírá nejen volbu programovacího jazyka, ale i otázky výuky algoritmizace, architektury softwaru a další problémy.

KLÍČOVÁ SLOVA:
Výuka IT, programování, algoritmizace, programovací jazyk, kurs programování, počítačová gramotnost

1. ÚVOD

Současná doba bývá charakterizována jako doba nástupu informačních technologií a přechodu k informační společnosti. Vzdor tomu je výuce informačních technologií u nás věnována poměrně malá pozornost – alespoň zkušenosti se studenty prvních ročníků tomu nasvědčují.

Je asi jasné, že na vysokých školách humanitního směru nelze od výuky informatiky očekávat více než seznámení s internetem a s textovými procesory, v lepším případě ještě s tabulkovými kalkulátory. Na druhé straně od absolventa vysoké školy technického směru lze očekávat poněkud hlubší znalosti v oblasti informačních technologií.

Zde si ovšem musíme položit několik otázek:

- Koho učíme?
- Jaké jsou jeho předchozí znalosti?
- Co on od nás očekává?
- Co je naším cílem?
- Co mu můžeme nabídnout?

V následujícím textu se pokusím na tyto otázky odpovědět. Moje odpovědi jsou ovlivněny zkušenostmi s cca 20 lety výuky v oboru IT na Fakultě jaderné a fyzikálně inženýrské ČVUT.

2. SOUČASNÝ STAV

V tomto oddílu se pokusím odpovědět na první dvě otázky.

Výsledky mého soukromého průzkumu mezi studenty, kteří nastupují do prvního ročníku FJFI ČVUT, ukazují, že v posledních cca 5 letech se 20—30 % studentů setkalo s programováním (i když mnozí do toho zahrnují i tvorbu webových stránek v některém z redakčních systémů). Dalších cca 40—50 % má zkušenosti s textovými a tabulkovými procesory a zbylí – přibližně 20—40 % – znají počítač víceméně jen jako nástroj pro hraní počítačových her, chat a „brouzdání“ po webu.

Znalosti těch, kteří již mají nějaké zkušenosti s programováním, se velmi liší. Studenti, kteří již skutečně programovali, zpravidla znají Visual Basic pro aplikace (z balíku MS Office), někteří znají základy PHP nebo jiného skriptovacího jazyka. Znalosti programovacích jazyků, jako je Java, C, nebo Pascal jsou spíše výjimečné.

Také postoj k informačním technologiím se ve studentské populaci v prvním ročníku značně liší.

- Část studentů je vnímá jako předmět svého zájmu a podle toho jim věnuje pozornost.

- Část studentů je vnímá jako nástroj, který sice není hlavním předmětem jejich zájmu, ale který budou potřebovat pro svou práci.
- Část studentů se na ně dívá jako na nutné zlo – buď proto, že na FJFI přišli studovat něco jiného, nebo prostě proto, že nejtěžší zkoušky v prvním ročníku jsou matematicky orientované předměty a díky tomu se předměty z oblasti informačních technologií ocitají na vedlejší koleji.
- Vzácně jsem se setkal i s případy studentů, kteří je odmítali jako nepotřebné. Poznamenejme, že svůj postoj většinou nedokázali racionálně vysvětlit, jednalo se spíše o něco jako politické přesvědčení (nebo možná provokaci).

Je ovšem jasné, že tyto postoje s v průběhu času mění a vyvíjejí a názory *úspěšných* absolventů se značně liší od názorů nastupujících studentů.

2.1 Co od nás student očekává

Pokud jde o programování, lze odpověď na třetí otázku podle mých zkušeností shrnout do následujících bodů:

- Student, který bere výuku IT jako nutné zlo, neočekává nic, snaží se splnit minimální požadavky a tím to pro něj končí.
- Zbývající studenti berou výuku IT, a především programování, ve značné míře jako něco, co jim zabezpečí dobře placenou práci v případě, že nedokončí úspěšně vysokou školu.
- Teprve pak uvažují někteří z nich o tom, co by vlastně měli znát, aby mohli být více než řekněme řadoví kodéři.

Tyto představy vedou k požadavkům, aby výuka programování začínala jazykem Java (neboť ten je v softwarových firmách požadován nejčastěji), k požadavku nezabývat se příliš kvalitou kódu, a když, tak jen na úrovni formálních automaticky testovaných metrik, kterým lze relativně snadno vyhovět (protože tak to dělají ve firmách) atd. Uvedené požadavky, někdy prosazované pomocí studentských anket, jsou ovšem často v přímém rozporu s očekávaným profilem vysokoškolsky vzdělaného odborníka.

Pokud jde o všeobecnou počítačovou gramotnost, očekávání se velmi různí. Někteří studenti přicházejí s představou, že je naučíme pracovat se všemi možnými „zajímavými a užitečnými“ programy, o nichž kdy slyšeli („A kdy nás budete učit Photoshop?“). Zpravidla ale očekávají, že se naučí principy práce s dokumenty – což je kupodivu velice rozumná představa.

Specializované předměty, jako je počítačová grafika, se učí až ve vyšších ročnících, a tam jediné, co většina studentů očekává (nebo to alespoň předstírá), je kvalitní výuka.

2.2 Cíle

Je jasné, že cílem výuky v oboru IT na vysoké škole technického směru musí být příprava odborníků se vzděláním odpovídajícím potřebám oboru, ve kterém budou pracovat. To ale znamená, že – pokud jde o IT – by absolvent se měl přinejmenším:

- Být seznámen s nejdůležitějšími programy používanými pro podporu práce v daném oboru, jako je např. Matlab pro běžné i méně běžné matematické výpočty nebo třeba knihovna GEANT pro simulace detektorů elementárních částic.
- Ovládat základy vytváření dokumentů typu vědeckých, technických nebo obchodních zpráv v textových a tabulkových procesorech a vytváření odpovídajících prezentací.
- Znát základy programování, neboť zdaleka ne všechny problémy jsou pokryty vhodnými komerčními nebo freewarovými aplikacemi, a i v případě, že pokryty jsou, je k nim často třeba umět napsat doplňky, které řeší specifické nestandardní problémy.

2.3 Cíle v případě softwarového zaměření

V případě studentů, jejichž zaměření se přímo týká vytváření softwaru, musí být cílem nejen znalost několika běžných programovacích jazyků a jejich knihoven, ale řada dalších věcí, z nichž zde uvedu tyto:

- Absolvent by měl znát běžně používané softwarové technologie, jako je vývoj řízený testy, refaktorování, používání návrhových vzorů atd. Požadavek znalosti OOP je dnes již samozřejmostí, student by ale měl mít možnost poznat i jiná paradigmata, jako je např. funkcionální programování.
- Absolvent by měl znát typické architektury aplikací v různých oblastech, nejen pokud jde o tvorbu obchodních aplikací. To znamená, že vedle řekněme vícevrstvých databázových aplikací s tenkým klientem by měl poznat architekturu aplikací pro řízení technologických procesů, architekturu aplikací pro sběr a zpracování dat z rozsáhlých zdrojů, architekturu operačních systémů atd.
- Absolvent by měl mít alespoň nákladní znalosti o algoritmech a jejich složitosti a umět odhadnout složitost programů, které vytváří.
- Absolvent by měl mít zkušenost s prací v týmu a s vedením týmu.
- Absolvent by měl umět komunikovat se zákazníkem – tedy umět zjistit jeho požadavky.

3. VÝUKA PROGRAMOVÁNÍ

Nyní se zaměříme na výuku programování. Nejprve si ale musíme ujasnit, co od úvodního kurzu programování – typicky jednosemestrálního – lze očekávat.

Zkušenosti, o nichž jsem hovořil v 2.1, nás nutí předpokládat, že studenti nemají žádné předběžné znalosti. Cílem pak je naučit je rozumět základním programovým konstrukcím a umět v nějakém programovacím jazyce samostatně napsat, přeložit a odladit program o složitosti cca několika set řádků.

Programovací jazyk pro úvodní kurz

Statistiky (viz např. [1], [2]) ukazují, že v současné době většina vysokých škol u nás i ve světě vyučuje jako první programovací jazyk Javu. Na druhém místě se podle týchž pramenů objevuje Visual Basic.

Co mluví pro Javu?

- Především poptávka softwarových firem a představa studentů, že její znalost postačí ke získání dobře placené práce.
- Základní nástroje pro vývoj v ní jsou k dispozici zdarma.
- Je to zdaleka nejpoužívanější programovací jazyk [3], obsahuje všechny konstrukce, které se v dnešních programovacích jazycích požadují, je zabezpečen proti chybám při práci s pamětí a je (téměř) čistě objektový.

Co mluví proti Javě?

I zde lze najít řadu argumentů.

- Jde o jazyk se syntaxí odvozenou od jazyka C, tedy pro začátečníka značně obtížnou.
- Z poznámek úspěšných absolventů [4] vysokých škol vyplývá, že volba Javy jako prvního jazyka vede někdy i k tomu, že studenty odradí od programování.
- Není vhodná pro všechny typy aplikací – lze si např. těžko představit, že v Javě vyvíjíme aplikaci pro řízení sběru dat nebo pro řízení technologického procesu.

Domnívám se, že zejména skutečnost, že Java může studenty odradit od programování, je důvodem, proč Javu nevolit jako jazyk pro vstupní kurz programování pro studenty s jiným než softwarovým zaměřením.

Vedle toho je – z hlediska úvodního kurzu – nevýhodou i skutečnost, že Java obsahuje prakticky všechny hlavní konstrukce, které se mohou v současných jazycích hlavního proudu ob-

jevit. To se týká např. výjimek nebo genericity. Úplný začátečník, který se na počátku potýká se základními pojmy, jako je proměnná nebo cyklus, jimi bude v jednosemestrálním úvodním kurzu zbytečně zahlcen. Na druhé straně ovšem očekává, že po absolvování tohoto kurzu bude umět slušně programovat – a jednoduchým měřítkem pro něj je, zda se naučil „celý jazyk".

Alternativy

Místo Javy lze uvažovat o použití dalších běžných jazyků, tedy o C, C++,C#, PHP, Basicu (Visual Basicu), JavaScriptu nebo Pascalu. Jinou možností je navrhnout a používat vlastní programovací jazyk přizpůsobený potřebám výuky.

Proti jazyku C mluví – stejně jako proti Javě – obtížnost jeho syntaxe pro začátečníka. Velice důležitým argumentem proti němu je také jeho liberalizmus – překladač jazyka C vždy hledá cestu, jak programátorovi vyhovět, takže některé konstrukce, které student nepochopí správně, se sice přeloží, protože jsou syntakticky správné, ale budou dělat něco jiného, než student očekává. Důsledkem bývá zdlouhavé ladění a někdy i odklon studenta od programování. *Cílem úvodního kurzu je naučit začátečníka přesnému myšlení a vyjadřování*, a pro to se hodí spíše restriktivní jazyky.

Dalším velmi silným důvodem proti jazyku C je, že není objektově orientovaný.

Proti jazyku C++ mluví stejné důvody jako proti jazyku C++ (s výjimkou neobjektovosti). Navíc k tomu přistupuje jeho značná komplikovanost (jde o jazyk o řád komplikovanější než je Java).

V podstatě stejné námitky lze vznést i proti jazyku C#.

Pokud jde o jazyk Basic, je situace složitější v tom, že na první pohled nemusí být jasné, o jakém jazyku vlastně hovoříme. O použití klasického – neobjektového a vlastně i nestrukturovaného – Basicu snad už dnes nikdo ani neuvažuje, takže by šlo spíše o implementaci tohoto jazyka od firmy Microsoft známou pod označením Visual Basic. Jeho starší verze (včetně VB pro aplikace, který byl používán pro programování maker v balíku MS Office) se podle mého názoru k výuce opět příliš nehodí, protože sice umožňují objekty používat, neumožňují je ale v plné míře definovat a využívat polymorfizmu.

Nevýhodou skriptovacích jazyků, jako je PHP nebo JavaScript, je jejich beztypovost. Domnívám se, datový typ je jedním ze základních pojmů, s nímž by se měl začátečník seznámit.

Poslední z jazyků, o kterém se zde zmíním, je Pascal. I zde je ovšem třeba vyjasnit, o jakou verzi jde. Tento jazyk byl na počátku 70. let navržen právě pro výuku programování; jeho syntaxe je snadno pochopitelná a je dostatečně restriktivní, takže řadu chyb odhalí již překladač. V současné době je ale ztotožňován buď s Object Pascalem v Delphi, nebo s Turbo Pascalem či Free Pascalem, což jsou vlastně jediné používané implementace.

Proti použití Object Pascalu z Delphi mluví opět zbytečná složitost, navíc se tento jazyk s každou další verzí dále rozšiřuje a košatí. Ještě horším problémem ovšem je, že není dobře dokumentován, takže při používání některých konstrukcí se student dostane do situace, kdy „programuje náhodou" – neví, proč mu program funguje a za jakých okolností fungovat nebude.

Turbo Pascal verze 6 nebo 7 (a podobně Free Pascal) je objektově orientovaný, i když neobsahuje takové konstrukce, jako jsou rozhraní nebo abstraktní třídy, bez nichž je výuka OOP přinejmenším problematická. Neobsahuje ovšem také výjimky a další konstrukce, které jsou v úvodním kurzu spíše na překážku.

Použití vlastního jazyka

I když to na první pohled vypadá podivně, není to ojedinělé. Ostatně např. programovací jazyk APL, populární na malých počítačích v 70. letech, byl původně navržen právě jako jazyk pro výuku informatiky. Při návrzích na použití něčeho podobného – např. jazyka C.C [5] –

jsem však narazil na odpor jak ze strany studentů, tak i ze strany kolegů a zdá se, že podobnou zkušenost mají i jiní.

Zdůvodnění bývají zpravidla typu „jiní to také nedělají," „moje přednáška na tvou navazuje" apod.

Chybí jazyk

Z uvedených poznatků jak se zdá vyplývá, že v současné době chybí široce akceptovaný programovací jazyk určený primárně pro úvodní kurs programování; proti všem používaným jazykům lze mít řadu námitek.

4. VÝUKA ALGORITMŮ

I když by se mohlo zdát, že algoritmizace je základem programování, lze se setkat s následujícími námitkami proti jejich výuce:

- Programátor je nepotřebuje. Vše je již v knihovnách.
- Programátor je nepotřebuje. Správný rozklad na třídy a objekty je důležitější a obstará vše potřebné.

Je pravda, že návrh a volba algoritmu je v softwarových firmách obvykle záležitostí analytika, nikoli řadového programátora. Lze ale předpokládat, že absolvent vysoké školy technického směru nebude chtít zůstat pouhým programátorem (kodérem). To znamená, že potřebuje mít mj. alespoň základní vědomosti algoritmech: Měl by znát běžné algoritmy, jako je quicksort nebo hešování, měl by ovládat dekompozici problému metodou shora dolů. Vedle toho by měl umět posoudit algoritmickou složitost navrženého postupu apod.

Příležitostní programátoři potřebují tyto vědomosti také. Připomeňme si, že do této kategorie patří většina absolventů technických vysokých škol, neboť jejich hlavním zaměstnáním je něco jiného než programování, a programy píší, pouze když potřebují vyřešit problém, na který nemají již hotový software. Ve skutečnosti jsou ve složitější situaci než běžný programátor v softwarové firmě, protože musejí zvládnout jak analýzu požadavků a návrh programu, tak i kódování, ladění a testování. Jejich jedinou výhodou je, že zpravidla sami rozumějí požadavkům, které jsou na jejich program kladeny.

Poznamenejme, že stejná situace nastává i v případě, že příležitostní programátoři píší doplňky k existujícím aplikacím – ať už jde o Matlab nebo o jiný program.

5. PSÁT CELÝ PROGRAM

Setkávám se také s požadavkem začínat výuku tím, že student bude připisovat metodu k existujícímu objektu. Jinak řečeno, neměli bychom začínat celým programem, ale jeho částí, jak je to typické pro práci v týmu v softwarových firmách.

V této souvislosti si nemohu nevzpomenout na dobu před nějakými třiceti lety, kdy zástupci podnikové sféry vystupovali na nejrůznějších fórech s příspěvky na téma „proč je na vysoké škole učíme derivovat, když pak nevědí, při jaké teplotě taje železo". Ponecháme-li stranou emoce, je jasné, že takto lze začínat výuku programátora na nejnižší úrovni, kodéra, ale ne vysokoškolsky vzdělaného člověka, který by měl od počátku rozumět souvislostem. Navíc je třeba mít stále na paměti, že většina absolventů technických vysokých škol bude patřit spíše k příležitostným programátorům, takže bude psát nejen doplňky k existujícím programům, ale i celé programy.

6. ZÁVĚR

Cílem mého příspěvku je vyvolat diskuzi o výuce programování na vysokých školách, zejména technického a přírodovědného zaměření. Přitom jsem záměrně pominul základní problém –

získání a udržení kvalitních lektorů s dostatečnými teoretickými znalostmi podloženými praktickou zkušeností.

Poděkování
Tento příspěvek vznikl v rámci grantu č. LA 08010 Ministerstva školství, mládeže a tělovýchovy České republiky.

LITERATURA

1. M. de Raadt, Richard Watson, Mark Toleman: *Introductory Programming: What's Happening Today and Will There Be Any Students to Teach Tomorrow?* In: Conferences in Research and Practice in Information Technology, Vol. 30. Eds R. Lister a A. Young. Australian Computer Society (2004). Viz též http://crpit.com/confpapers/CRPITV30deRaadt.pdf
2. M., de Raadt, R. Watson a M. Toleman. *Language Trends in Introductory Programming Courses*. The Proceedings of Informing Science and IT Education Conference, Cork, Ireland, InformingScience.org. (2002)
3. Index TIOBE. http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html (březen 2009)
4. *Curricula Recommendations*. Association for Computing Machinery. http://www.acm.org/education/curricula-recommendations
5. Merunka, V., Nouza, O., Virius, M.: *Programovací jazyk C.C*. Automatizace 9 (2008), str. 562—565.