PROGRAMOVÁNÍ JE TAKÉ LADĚNÍ

Rudolf Pecinovský
ICZ a.s., Na hřebenech II 1718/10, 140 00 Praha 4,
VŠE Praha, Fakulta informatiky a statistiky, Katedra informačních technologií
rudolf@pecinovsky.cz

ABSTRAKT:

Hovoříme-li někdy o výuce programování, rozebíráme většinou výuku kódování, doplněnou v lepším případě o výuku zásad analýzy a návrhu programů. Zapomínáme přitom na to, že většinu času netráví studenti (a stejně tak i profesionální programátoři) navrhováním programu a psaním kódu, ale laděním. Ladění programu se však ve výuce nevěnuje žádná, anebo v lepším případě jen minimální pozornost. Příspěvek se zabývá možnostmi začlenění výuky zásad správného ladění a testování do celkové koncepce výuky programování.

KLÍČOVÁ SLOVA:

Výuka programování, ladění, debugging.

1 ÚVOD

Většina učebnic, které se vydávají za učebnice programování, se věnuje převážně syntaxi probíraného jazyka a vysvětlování obsahu dostupných knihoven. Tu tam se najde i učebnice, která se snaží naučit i to, jak programy navrhovat. Tím, jak v programech vyhledávat chyby a jak je následně odstraňovat, se autoři zabývají jen výjimečně a pokud ano, tak pouze na několika málo stránkách, přičemž zde jsou uváděna pouze nejobecnější pravidla bez konkrétních příkladů, na nichž by si student mohl vyzkoušet, jak látku pochopil obdobně, jako je tomu u většiny ostatních probíraných témat. Čtenáři pak získávají dojem, že programátoři prostě program napíší, spustí, a program chodí.

O to větší je pak jejich rozčarování, když zjistí, že jejich programy nejsou tak dokonalé, jak při čtení učebnic či sledování výkladu pochopí, že by měly být. Nejvíce je pak frustruje jejich neschopnost chybu odhalit a opravit. Ta je často ještě posilována vyučujícím, který v časové tísni pouze studentům ukáže, kde mají v programu chybu, či v horším případě tuto chybu studentům rovnou opraví. Student si pak nemůže vzít z této opravy žádné poučení.

Řada vyučujících tvrdí, že učí studenty hledat a opravovat chyby. Podíváte-li se ale zblízka na obsah této výuky, zjistíte, že se studenti pouze učí, jak psát testovací programy, pomocí nichž lze odhalit předem očekávatelné chyby. Co se má přesně testovat, to většinou určí učitel. V učebnicích ani kurzech, s nimiž jsem se mohl setkat, se ale nikde neučilo odhalování a opravování neočekávaných chyb. V e svém příspěvku bych se chtěl zamyslet nad tím, jestli není možno tuto praxi změnit.

V následujících pasážích nejprve naznačíme typické situace, s nimiž se setkáváme u studentů, kteří při psaní programu narazí na chybu. (S obdobnými situacemi se většina vyučujících již mnohokrát setkala a možná by doplnila i další.) Poté se pokusím zamyslet, co vše bychom potřebovali pro to, abychom mohli studenty učit programy nejenom navrhovat, ale také ladit.

2 SKUPINY STUDENTŮ

Každý máme své zkušenosti se studenty, kteří bloudí svými programy a nejsou v nich schopni odhalit chybu. Před tím, než začneme uvažovat nad tím, jak studenty naučit ladit programy, bychom si měli připomenout, že studenti programování většinou nejsou homogenní skupinou. Mohli bychom je rozdělit např. následovně:

- Studenti, kteří nestudují programování o své vůli. Chtějí studovat zcela jiný obor (většinou management) a programování je pro ně jako povinný předmět jenom nutné zlo.
- Studenti, které programování do jisté míry zaujalo a chtějí se je naučit. Ty opět rozdělíme do dvou skupin:
    - Začátečníci, kteří v životě neprogramovali.
    - Ti, kteří již umějí programovat a přinášejí si s sebou návyky, z nichž mnohé jsou vysloveně špatné.

Každá skupina potřebuje specifický přístup. Podívejme se na některé typické situace a reakce, s nimiž se můžeme setkat napříč skupinami. snad s výjimkou těch nejzkušenějších:

2.1 Ve vašem programu je chyba!

Snad nejčastější situací je ta, kdy student ohlásí, že v programu, o němž tvrdí, že jej psal přesně podle návodu (v mnoha případech dokonce opsal z promítané prezentace či přímo zdrojového kódu), je chyba. Když se zeptáte jaká, neví – prostě chyba. On jej nevymýšlel, tak přeci nemůže vědět, kde ta chyba je. Zejména v počátečních hodinách patří tato situace k nejčastějším.

Všichni studenti jsou si vždy naprosto jisti tím, že jejich program je do písmene shodný se vzorovým. Nějaký překlep si nepřipouští a udivuje mne, že je to nenapadne, přestože vidí, že jejich kolegům stejný program chodí. Je opravdu s podivem, že se prakticky žádný student nepokusí přečíst obdržené hlášení, které často jasně naznačuje, v čem by chyba mohla být. Začátečníci sice nemusí v prvních hodinách tuto nám zcela jasnou zprávu pochopit, ale měli by se o to alespoň pokusit. Když se jim totiž stejná zpráva objeví na displeji příště, budou již stavět na své předchozí zkušenosti.

Chyby vzniklé při opisování nějakých ukázkových kódů bývají většinou syntaktické chyby – tj. něco, co profesionální programátor vlastně za chybu ani nepovažuje. Hlášení překladače pak bývají v takovýchto situacích dostatečně návodná, aby podle nich mohl i relativní začátečník chybu objevit.

2.2 Hlásí to nějakou chybu

Obdobná situace nastává v okamžiku, kdy se chyba objeví v programu, který sami napsali. Studenti jsou naprosto přesvědčeni, že chyba nemůže být v tom jejich malinkém prográmku, ale musí být někde v té „obludě“, která jejich program přeloží a spustí.

Pravda, ohlásí-li se chyba v programu vyhozením nějaké výjimky, bývá tato zpráva na první pohled ještě méně srozumitelná, než hlášení překladače o syntaktické chybě. Nicméně i hlášení výjimky se dá v řadě případů poměrně jednoduše analyzovat a zdroj problémů detekovat.

Nejdůležitější ale je, aby studenti přijali za svou zásadu, že čím je člověk větší začátečník, tím dříve hledá chybu v překladači, vývojovém prostředí, operačním systému či kdekoliv jinde, jen ne u sebe. Vysvětluji jim, že za těch více než asi 35 let, co programuje, se jenom třikrát stalo, že chyba nebyla v mém programu. V průměru to tedy vypadá, že intenzivně pracující programátor na takovou chybu narazí jednou za 10 let. A musí se k tomu pohybovat v oblastech, kam běžní programátoři nezabrousí, protože jinak by se na tu skrytou chybu už dávno přišlo a byla by opravena. Školní úlohy se však pohybují v oblasti natolik probádané, že výskyt chyby v některém z podpůrných programů opravdu nehrozí.

2.3 Počítač dělá něco jiného, než co je napsané v programu

Další typickou situací je chování programu, které naprosto odporuje zadanému kódu. Studenti mi ve svém kódu ukazují pasáž, ve které do nějaké proměnné cosi přiřadí, aby pak počítač

tvrdil, že v dané proměnné nic není. Počítač podle nich vykonává jejich program špatně, tj. dělá něco jiného, než co oni naprogramovali.

Je to opět variace dříve popsaného jednání: chyba není u mne, chyba je v počítači. Moje první otázka v takovýchto situacích bývá: „Podle čeho usuzujete, že počítač touto částí programu prošel?“ Neustále je musím přesvědčovat: „Nevěřte ničemu, co jenom vidíte v kódu. Věřte jenom tomu, co vám počítač doopravdy ukáže. Ověřte si v takovýchto okamžicích nejprve to, že počítač při vykonávání programu do inkriminované oblasti vůbec došel.“

2.4 Shrnutí

Výše popsané situace jsou těmi nejjednoduššími. Jejich řešení jsou většinou jednoduchá a nevyžadují žádné hluboké znalosti ladění. Většinou opravdu stačí, aby se student vnitřně ukáznil, četl obdržené zprávy a hledal chyby jen ve svých programech. Opravdové ladění nastupuje až v okamžiku, kdy jsou jednoduchá řešení nepoužitelná a kdy pak studenti tráví hodiny prohlížením svého kódu, jeho porovnáváním s ukázkovými řešeními a marným bádáním nad tím, kde by mohla být tentokrát ukryta chyba.

3 JAK UČIT LADĚNÍ

Začnu z jiného soudku. Většina matematiky na základních a středních školách je pouze o dosazování do vzorečku. Výjimkou jsou pouze konstruktivní úlohy v geometrii a úprava algebraických výrazů. Pro tyto dvě oblasti neexistuje obecné pravidlo, jak zadanou úlohu vyřešit. Zkušenost ukázala, že když student (byť s návodem a dopomocí) vyřeší dostatečný počet příkladů, dokáže další příklady řešit sám.

Obdobné je to i při výuce programování. Existuje řada problémů, jejichž řešení můžeme „degenerovat“ na prosté „dosazení do vzorečku“:

- Objeví-li se v programu místo, odkud mohu pokračovat několika způsoby, musím nejprve vymyslet podmínku, podle které poznám, která z cest je ta správná, a pak použít podmíněný příkaz.
- Má-li opakovaně provádět něco s prvky pole, použiji cyklus.
- Objevuje-li se nějaká část kódu na více místech, definuji ji jako podprogram (proceduru nebo funkci) a z těchto míst ji už jenom volám.
- …

A tak bych mohl pokračovat ještě dlouho. A to nepočítám architektonické a návrhové vzory, které také mají jakási pravidla, kdy je vhodné je použít, i když se při jejich implementaci pohybuji ve výrazně vyšší hladině abstrakce.

Vedle toho existuje řada úloh, pro jejichž řešení není možné definovat takováto pravidla (i když existence stále rozsáhlejších knihoven a frameworků jejich množství stále snižuje) a musíme proto postupovat obdobně, jako postupují matematici při výuce upravování výrazů či řešení konstruktivních úloh.

Podobně je to i s laděním. Existuje několik pravidel, jak postupovat v typických situacích, např. vyhazuje-li program výjimku `NullPointerException`, jenomže tato pravidla nás většinou nedovedou až k cíli, ale pouze naznačí doporučený směr.

3.1 Literatura

Jak jsem se již zmínil na začátku, většina knih zabývajících se programováním a zejména pak knih, které se snaží být vstupními branami do tohoto světa, se o ladění zmiňuje jen velmi letmo, a pokud se rozhodnou věnovat této problematice podrobněji, zůstanou u obecných tvrzení typu: „definujte pro každou třídu její testovací třídu“, „prokládejte kritická místa programu kontrolními tisky“, „využívejte příkazu assert“ apod.

Existuje několik knih zabývajících se problematikou ladění vyvíjených programů, avšak všechny jsou určeny pro čtenáře s jistými zkušenostmi. Čtenáři, kteří se teprve začínají učit programovat, je budou číst jen s obtížemi. Kdo se však che v této oblasti vzdělat a je ochoten tomu věnovat jisté úsilí, má možnosti.

3.2 Sbírky příkladů

Když vzpomenete na to, jak jste se učili matematiku, jistě se vám vybaví sbírky příkladů, které bylo třeba vypočítat, aby se váš mozek naučil nějaké úlohy řešit. Sbírky příkladů existují i pro výuku programování – mnohé učebnice a kurzy jsou na příkladech přímo postaveny, takže při jejich studiu máte bohatou studnici vzorových řešení, z nichž se můžete poučit na dobu, až budete vyvíjet řešení vlastní.

Na druhou stranu o žádné sbírce příkladů, na nichž byste si mohli nacvičit dovednost ladit programy, nevím. Určitě by bylo vhodné nějakou takovou sbírku sestavit a nabídnou začátečníkům.

Jedním takovým pokusem o „minisbírku“ určenou právě naprostým začátečníkům je práce [9]. Autor v ní předvádí několik jednoduchých chyb spolu s postupem, jak danou chybu odhalit a odstranit. Práce je doplněna sadou animovaných příkladů, které se snaží ukázat proces ladění „naživo“.

Zmíněné práci by bylo lze určitě leccos vytknout. Mnohé lze omluvit autorovu nezkušeností, něco jde na vrub pravděpodobného závěrečného spěchu, s nímž se takovéto práce odevzdávají. Nicméně práce naznačuje, kam by bylo vhodné napřít své úsilí.

4 ZÁVĚR

Příspěvek se zaměřil na dosud opomíjenou oblast výuky programování, kterou je výuka ladění. Shrnul současný stav a vypíchl nedostatečnou výuku této dovednosti ve vstupních kurzech programování. Poté připomenul několik nejčastěji se vyskytujících problémů, s nimiž se u studentů při ladění programů setkáváme spolu s doporučeními, jak studenty správně nasměrovat. Na závěr seznámil s jednou studentskou prací snažící se řešit nedostatek vhodných výukových materiálů v této oblasti a naznačil směr dalších prací.

LITERATURA

[1] PECINOVSKÝ Rudolf: *Metodika Design Patterns First – Jak lze drobnou změnou výkladu zlepšit pochopení některých objektových konstrukcí*. Sborník konferece DidInfo 2010. ISBN 978-80-8083-952-9.

[2] PECINOVSKÝ Rudolf: *OOP – Naučte se myslet a programovat objektově.* Computer Press 2010. ISBN 978-80-251-2126-9.

[3] PECINOVSKÝ Rudolf: *Skvělý program nemusí být skvěle prodejný*. Sborník konference Objekty 2007.

[4] PECINOVSKÝ Rudolf: *Výuka OOP žáků základních a středních škol*. Sborník konference Objekty 2003, Ostrava, ISBN 80-248-0274-0.

[5] PECINOVSKÝ Rudolf: *Proč a jak učit OOP žáky základních a středních škol*. Žilinská didaktická konferencia, 2004, Žilina.

[6] PECINOVSKÝ Rudolf: *Výuka programování pro praxi*. Sborník konference Informatika XXI 2008, Lázně Luhačovice.

[7] PECINOVSKÝ Rudolf: *Mýty ve výuce programování a metodika Design Patterns First* – zvaná přednáška. Sborník konference Objekty 2008, Žilina.

[8] TVRDÍK Josef: *Modulárně nebo strukturovaně?* Seminář ***Programování '85*** – ke stažení na http://formular-ekf.vsb.cz/formulare/F01/tsw/?page=prispevek&rok=1985&prispevekid=262.

[9] ZÁVĚRKA Jakub: *Alternativní výukové materiály pro vstupní kurzy programování* – Bakalářská práce, VŠE 2010.