Automatizace vyhodnocování studentských úloh

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 Problémy s vyhodnocováním odevzdaných řešení

Příprava zadání studentských úloha a zejména pak jejich následné vyhodnocování patří pro mnohé z nás k těm méně příjemným složkám výukového procesu. Budou li mít všichni studenti shodné zadání, hrozí nebezpečí, že si navzájem předají jeho řešení, které pak ti bystřejší z nich ještě několika hromadnými záměnami formálně modifikují, aby shodnost programů nebila tak do očí. Budou li mít různá zadání, přidělláváme si práci s jejich následných vyhodnocováním.

Někteří vyučující se proto snaží vymýšlet úlohy, které se budou následně dobře vyhodnocovat, ale tato snaha většinou končí u toho, že úloha někam vypíše jakési hodnoty, které pak „zkontrolujeme očima“ nebo o jejich kontrolu požádáme nějaký program, který tyto hromadně zkontroluje výsledky. Problém je v tom, že při klasických podobách zadání, která známe z nejrůznějších učebnic a kurzů, je takováto automatizace rozumně řešitelná pouze u zadání vyžadujících vytvoření jednoho zdrojového souboru s programem používajícího pouze standardní vstup a výstup.

Jakmile nás nebude zajímat pouhý výsledek, ale chtít budeme po studentech něco složitějšího, či jakmile se dokonce budeme chtít dozvědět něco o struktuře testovaného programu, začneme mít s automatizací vyhodnocování odevzdaných řešení problémy. Jedinou možností u klasicky koncipovaných překládaných jazyků je sáhnout po složitých nástrojích, které analyzují zdrojový kód nebo realizují některé z funkcí ladících programů.

Modernější jazyky postavené nad platformami Java a .NET jsou koncipovány tak, že mnohé z těchto informací můžeme zjistit i za chodu programu. Jejich první výhodou je, že překládají program do mezikódu, jenž stále obsahuje řadu informací o zdrojovém kódu. Druhou mocnou zbraní je možnost využití reflexe, kterou obě platformy podporují měrou vrchovatou. Analýza programu prostřednictvím reflexe je mnohem dokonalejší než analýza nabízená klasickými prostředími. Navíc ji lze mnohem snáze naprogramovat. Stále je však mnohem pracnější než obdobná analýza, která může využít přímé znalosti o rozhraní programu, jež lze navíc deklarovat přímo v zadání.

Všichni známe obdivuhodnou studentskou schopnost porušovat nejrůznější zadané konvence. Tato porušení však většinou nebývají záměrná, ale jsou pouze výsledkem nedostatečného soustředění se při výkladu nebo čtení zadání řešené úlohy. Když studentům nabídneme nástroj, který za ně tato porušení zkontroluje a upozorní je na ně, procento úspěšných řešení se výrazně zvýší.

2 Prostředek řešící mnohé problémy: `interface`

Jednou z programových konstrukcí, které nám mohou plnit úlohu takovéhoto „výchovného prostředku“ je `interface` – konstrukce, jejímž prostřednictvím můžeme zadat své požadavky na signaturu rozhraní definované třídy tak, aby ji mohl překladač zkontrolovat. Mohli bychom tedy prohlásit, že `interface` je formálním zápisem signatury dané třídy. Používán konstrukce `interface` nám navíc dramaticky zjednoduší naprogramování testů kontrolujících správnost řešení zadané úlohy (viz [3], [4]).

Nevýhodou mnoha výukových koncepcí ale je, že tuto konstrukci přednášejí většinou až na konci základního kurzu – dost často na konci druhého semestru. Vynechám-li pedagogickou nevhodnost takovéhoto přístupu, pak jeho paralelní nevýhodou je i nemožnost využít jej k výraznému zefektivnění a zjednodušení kontroly odevzdávaných studentských prací.

Metodika *Design Patterns First* ([1], [5], [6], [7]) zařazuje výklad této konstrukce již na počátku výuky, na třetí cvičení (v prvním se interaktivně pracuje s objekty, v druhém studenti napíší svůj první zdrojový kód). Tím nám dává do rukou prostředky, které nám umožní pracovat se studenty prakticky od počátku výuky stylem, s nímž se setkají ve své budoucí praxi. Jako příjemný vedlejší efekt tak navíc získám nástroj, který nám umožní relativně snadno automatizovat i řešení poměrně složitých zadání.

3 Knihovny xUnit a jejich služby

Dalším nástrojem, který nám může při automatizaci vyhodnocování výrazně pomoci a současně díky včasné zpětné vazbě zvýšit kvalitu odevzdávaných prací, jsou knihovny pro automatické testování řady `xUnit` případně některé jejich odvozeniny. Tyto knihovny existují pro každý moderní programovací jazyk a platformu a vývojová prostředí, která nechtějí zapadnout na „smetiště dějin“, mají integrovánu jejich podporu.

Díky této zabudované podpoře můžeme studentům předat testovací třídu jako součást zadání a jak budou studenti postupně budovat své řešení, mohou si kontrolovat, jak postupně

procházejí zadané testy. Student, který nemá zpětnou vazbu od zhavarovaných jednotkových textů mnohem snáze podlehne pokušení pokládat svůj polotovar za hotový program. Když mu ale na obrazovce neustále svítí upozornění, že tato funkce ještě není plnohodnotně definována, většinou „vyměkne“ a pokusí se jí uvést do stavu, v němž testy projdou.

Zkušenost znovu ukazuje, že v řadě případů není „lenost studentů“ hlavní příčinou nekvalitnosti řešení. Oni by sice rádi byli s programem co nejdříve hotovi, ale když jim dodané testy nedovolí považovat program za dokončený, přemohou se a snaží se jej upravit tak, aby testům vyhověl. Abychom dále zvýšili jejich motivaci, vyhlašujeme na kroužku heslo:

Program, který skoro chodí, je jako letadlo, které skoro létá!

Dokud studenti nemají jednoznačný ukazatel funkčnosti programu, snaží se obhájit svůj program tvrzením, že si neuvědomili, že ta či ona funkce musí pracovat zadaným způsobem. Předem známé testy jim tyto výmluvy seberou.

4 Opsaná či objednaná řešení

Nyní na chvíli odbočím od původního tématu a dotknu se otázky přebírání řešení od kolegů, resp. objednávání si hotových řešení od přátel či dokonce za úplatu. Přiznejme si, že zabránit studentům přebírat cizí řešení, je poměrně obtížné a u domácích úkolů prakticky nemožné. Zdá se mi proto výhodnější se s touto možností smířit a upravit způsob odevzdávání úloh tak, aby přiměl studenty si odevzdávané řešení alespoň prostudovat.

Já to řeším tak, že veřejně vyhlásím, že je mi jedno, jestli student odevzdávané řešení vytvoří nebo si je nechá vytvořit, ale budu po něm chtít, aby se v odevzdaném řešení vyznal tak dobře, jako kdyby je napsal sám. Student proto musí odevzdané řešení obhájit. Při odevzdávání na něj čeká nějaký drobný úkol jak odevzdávanou úlohu modifikovat a teprve správně modifikovaná úloha je považována za odevzdanou.

Zkušenost ukázala, že řada studentů se domnívá, že když si nechá program udělat a z vysvětlování autora získá pocit, že pochopila jeho funkci, tak že pak budou schopni program samostatně upravit. Všichni ale víme, že tomu tak není. První pár semestrů se vyskytlo několik studentů, kteří přinesli vypracovanou úlohu, a když jsem jim drobně upravil zadání nebo naopak pokazil odevzdané řešení, s údivem zjistili, že program, o němž se ještě včera večer domnívali, že mu rozumějí, je pro ně nyní záhadným bludištěm, které nejsou schopni modifikovat ani opravit. Toto zjištění se mezi studenty velmi rychle rozšířilo, takže poslední dobou se již nepřipravení studenti prakticky nevyskytují.

5 Hromadné zpracování

Prozatím jsem hovořil o tom, jak zautomatizovat vyhodnocení jednoho programu. Základním problémem vyhodnocování studentských úloh je ale jejich množství. K zvládnutí množství vyhodnocovaných úloh používáme knihovnu, která poskytuje prostředky k nalezení všech programů, které řeší dané zadání, jejich vyhodnocení a přípravě výstupu, který poskytuje potřebné informace jak pro studenty, tak pro učitele.

Prozatím program pracuje tak, že obdrží `interface`, který mají vyhodnocovaná řešení implementovat, případně jinou charakteristiku vyhodnocovaných tříd. Vyhledá všechny třídy vyhovující zadané charakteristice a na každou implementuje předem zadaný test. Výsledek prozatím ukládá ve formě textového souboru, v němž jsou vypsány všechny odhalené prohřešky. Tento soubor si mohou studenti prohlédnout a pak přijít s otázkami na věci, které jim nejsou jasné. Učitel pak může ze souboru vyčíst nejčastější chyby a vrátit se k nim v příštím výkladu.

Ve [3] byla zmíněna první verze této knihovny. V současné době používám rozšířenou verzi knihovny poskytující vedle vyhledání tříd i podporu nejčastěji zadávaných tesů.

6 Závěr

Příspěvek připomněl problémy spojené s vyhodnocováním studentských úloh snahami o jejich automatizaci. Ukázal, že vhodné využití rozhraní (interface) nám dává do rukou velice mocný nástroj pro zadávání, ale zejména pak pro automatizované vyhodnocování samostatně řešených studentských úloh. Předvedl, jak je možno zadávat studentské úlohy tak, aby si studenti při jejich řešení maximálně osvojili návyky, které budou ve své následující praxi potřebovat, a současně tak, aby si vyučující maximálně ulehčil vyhodnocování odevzdaných řešení. Současně vysvětlil, jak použití metodiky *Design Patterns First* automatizaci vyhodnocování úloh výrazně usnadňuje.

7 Reference

[1] PECINOVSKÝ Rudolf: *Myslíme objektově v jazyku Java – kompletní učebnice pro začátečníky, 2. aktualizované a rozšířené vydání*, Grada, 2008. ISBN 978-80-247-2653-3.

[2] PECINOVSKÝ Rudolf: *Návrhové vzory*, CPress, 2007. ISBN 978-80-251-1582-4.

[3] PECINOVSKÝ Rudolf: Metodika *Design Patterns First* a vyhodnocování studentských úloh. Sborník konference Tvorba softwaru 2007, Ostrava. ISBN 978-80-248-1427-8.

[4] PECINOVSKÝ R., PAVLÍČKOVÁ J., PAVLÍČEK L.: Order of explanation should be Interface – Abstract classes – Overriding. *Proceedings of 12th Annual Conference on Innovation and Technology in Computer Science Education (ITiCSE'2007).* Dundee, ACM Press.

[5] PECINOVSKÝ, Rudolf: Aplikace metodiky Design Patterns Fist. *Objekty 2006 – sborník příspěvků desátého ročníku konference*, ČZU, Praha 2006. ISBN 80-213-1568-7.

[6] PECINOVSKÝ Rudolf: Výuka programování podle metodiky Design Patterns First. *Tvorba softwaru 2006 – sborník přednášek*. ISBN 80-248-1082-4.

[7] 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.

Abstrakt

Vyhodnocování studentských úloh je činnost nepříjemná a nudná. Její větší automatizaci brání mimo jiné i používané metodiky výuky programování. Pokud se proto na školách automatizuje vyhodnocení úloh, omezují se tyto snahy většinou na ověření správnosti výsledku. Vyhodnocení správnosti struktury programu a činnosti jeho jednotlivých částí se automatizuje jen výjimečně.

Příspěvek připomíná, že aplikace metodiky *Design Patterns First* ([1], [5], [6], [7]) automatizaci vyhodnocování výrazně usnadňuje ([3], [4]). Včasné zařazení výkladu konstrukce `interface` umožňuje definovat zadání úlohy takovým způsobem, abychom pak mohli velice snadno vyhodnotit funkčnost jednotlivých částí programu. Příspěvek dále připomíná některé základní skutečnosti ovlivňující přístup studentů k vypracovávání zadaných úloh. Ukazuje některé přístupy, které zvyšují ochotu studentů vytvářet domácí úkoly, zlepšují kvalitu odevzdaných prací a.usnadňují automatizaci vyhodnocování odevzdaných řešení. V další části se pak zabývá řešením problému plagiátorství kupování vyhotovení prací. V závěru pak seznamuje s knihovnou, která výrazně usnadňuje tvorbu programů, které automaticky vyhodnotí odevzdaná studentská řešení.

Přednesené postupy výrazně zvýšily kvalitu výukového procesu. Automatizace vyhodnocení úloh šetří neproduktivně strávený čas učitele a současně motivuje studenty. Programy, které studenti odevzdávají, obsahují méně chyb a případy, kdy student není schopen dále vylepšit odevzdaný program se již vyskytují spíš výjimečně.

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

Annotation

One of the bothersome tasks in programming education is evaluation of student assignment solutions and homeworks. Automation of this task is difficult due the used methodology of teaching. When we meet an attempt to automate the evaluation, it is mostly restricted to verifying the result of solution. Evaluation of the program structure and functionality of its particular parts is automated seldom.

The paper reminds that application of methodology *Design Patterns First* ([1], [5], [6], [7]) makes this task much simpler ([3], [4]). The early explanation of `interface` allows to de-

fine the assignment in the way allowing easy future automation of the evaluation of functionality of particular parts. The paper then reminds some basic facts which influent the students' approach to developing the assigned programs. It shows some attitudes increasing the students' willingness to solve these tasks, better the quality of the handed in solutions and easier the automation of evaluation of the handed in programs. In the next part it discuss plagiarism and buying the development of the solution. At the end it introduces the library, which significantly makes development of programs automating the evaluation much easier.

The presented solutions significantly enhances the quality of education. The automation saves the unproductive teacher time and simultaneously it motivates students. The gave in solutions contains much less errors and the situation, when a student is not able make some further enhancement of the given in program, are now only rare.