МАТЕМАТИКА И МАТЕМАТИЧЕСКО ОБРАЗОВАНИЕ, 2010
MATHEMATICS AND EDUCATION IN MATHEMATICS, 2010
Proceedings of the Thirty Ninth Spring Conference of
the Union of Bulgarian Mathematicians
Albena, April 6–10, 2010

АРГУМЕНТИРАЩО МИСЛЕНЕ В ОБЕКТНО-ОРИЕНТИРАНОТО ПРОГРАМИРАНЕ ПРИ ИЗГРАЖДАНЕТО НА ФУНДАМЕНТАЛНИ ЗНАНИЯ ЗА ПРОГРАМИРАНЕТО

Кирил Иванов

Формулирани са четири критерия, по които може да се преценява степента на усвояване на фундаменталните знания по програмиране във ВУЗ. Посочена е спецификата на аргументиращото мислене в обектно-ориентираното програмиране и ролята му за фундаменталната подготовка на студентите. Разграничени са два начина на разсъждаване – с единствен възможен резултат и с избор между няколко възможности при търсенето на баланс между противоречиви изисквания. Приведени са примери за аргументи, които значително подпомагат студентите в разбирането на фундаменталните идеи при използването на обекти и класове в различни етапи от изграждането на софтуерна система. Специално внимание е отделено на влиянието на аргументиращото мислене върху мотивацията на учащите, а оттам – и върху фундаменталната им подготовка.

1. Въведение. Всяко обучение на бакалаври и магистри по необходимост се съсредоточава върху фундаментални знания. От гледна точка на начина на обучение е още по-важно, че тъкмо най-дълбоките, основополагащите представи се оказват плодоносната "почва", върху която най-резултатно, стабилно и бързо се обогатяват знанията и уменията на студентите.

Обаче при обучението по програмиране, фундаменталните знания от много години насам вече са и единственото, което въобще би могло да подготвя пълноценни специалисти. Например, каквито и версии на език и среда за програмиране да са използвани в процеса на обучението, да се познават само те не стига. Тези версии със сигурност ще са неподходящи за редица задачи и цели (включително при кандидатстване за работа в някои фирми), а към момента на дипломиране на обучаемите, най-вероятно вече съществено ще изостават от множество актуални потребности.

2. Критерии за преценка на придобитите фундаментални знания и влияещите върху тях фактори в учебния процес. Както понятието *фундаментални знания*, така и анализът на усвояването и преподаването им се отличават със значителна абстрактност, сложност, относителност, многостранност и изненадващо разнообразни връзки с най-различни области. Затова, разкриването на подходящи фактори за успешното изграждане на такива знания може да става по много начини.

Една възможност е да потърсим характеристики на резултатите от обучението, които надеждно, убедително показват, че фундаменталните знания за програмирането са усвоени в приемлива, достатъчна степен. Когато такива характеристики са

закономерно следствие от някакви фактори, тогава последните могат да бъдат средство за изграждане на желания фундамент, а доколко са необходими или достатъчни за тази цел, определят зависимостите между исканите резултати и съответните фактори.

Критерии за нивото, до което обучаемите са овладели желаните фундаментални знания за програмирането, могат да бъдат следните умения:

У.1. Осъзнаването и разбирането на идеите и концепциите, определящи типичния за всяка парадигма на програмиране набор от възможности, предоставян във всички широко използвани съвременни езици и системи за програмиране, представители на парадигмата.

У.2. Умението *бързо* и *леко* да се усвоява или изучава нов език или система за програмиране.

У.3. Способността да се отгатват вече съществуващи, но все още неизучени средства на даден език или система за програмиране и да се предвиждат новите идеи, които в близко бъдеще ще получат вградена поддръжка в програмните езици или в технологичните средства за производство на софтуер.

У.4. Умението творчески и в срок да се създава адекватен софтуер с висока степен на оригиналност и сложност.

Четирите умения се намират в отношение на градация, в смисъл, че всяко от тях съществено използва и се опира на предходните, но заедно с това включва и нещо качествено ново, нещо съществено в повече спрямо тях. Така У.4 се явява кулминация, която предполага наличието на У.1, У.2 и У.3.

Отсъствието на каквито и да било прояви на изброените способности със сигурност показва и липсата на фундаментална подготовка, защото всяко основополагащо знание едновременно и развива, и изисква някакъв елемент от горните умения. Заради тази взаимна обусловеност и обвързаност, изброените аспекти на резултатите от обучението по програмиране представляват и показатели за наличието или отсъствието на фундаментални знания.

Нещо повече – всяко задълбочаване на познанията за основополагащите концепции, за ключовите идеи в света на програмирането закономерно разширява и засилва горните способности и обратно – всеки съществен прогрес в което и да било от четирите умения неминуемо води до по-пълни и по-стабилни фундаментални знания.

При това, изброените четири критерии отразяват много високи изисквания. Наличието в пълна степен и на четирите вида способности очевидно означава най-висша степен на професионализъм, до който дори изтъкнатите специалисти понякога само се доближават. Затова тези признаци са единствено ориентир, хоризонт, желан максимум на резултатите от обучението, чието достигане обаче (едва ли е възможно и) не е и не бива да бъде цел на висшето образование. Само отделни студенти успяват да развият в значителна степен в относително комплексен вид способности и от четирите типа. Това са именно най-изявените отличници, личностите с ярко изразена творческа нагласа, с много силна мотивация в програмирането или в близка до него област и, най-често, с ниво на подготовка (при постъпването във ВУЗ) далеч надхвърлящо училищния курс.

В същото време, дори учащите, отнасяни по всеобщо мнение към средното ниво на успеваемост, овладяват отделни умения от всеки от изброените видове, дори

от четвъртия. С други думи, в някаква тясна област, обикновено предизвикваща повишен интерес у студента, той съзнава дали е в състояние за смислено време да реализира нетипичен или сложен проект с творческо прилагане на най-важните глобални концепции в програмирането.

Следователно, изброените критерии могат да бъдат своеобразна, макар и сложна, многоаспектна и относително многозначна мярка за обхвата и дълбочината на придобитите фундаментални знания. По такъв начин, при фундаменталната подготовка на студентите значението на всеки отделен фактор в обучението може да бъде приблизително оценявано по степента, в която той развива съвкупността от горните четири типа умения.

3. Значение на аргументиращото мислене в обектно-ориентираното програмиране за фундаменталната подготовка по програмиране. Всеизвестна и постоянно потвърждавана от практиката истина е, че най-ценните и най-напредналите знания и умения се градят върху разбирането на изучавания предмет, а самото разбиране се крепи главно върху аргументиращото мислене. За болшинството специалисти, в която и да било област, пък и не само за тях, "разбирам" обикновено означава "мога да аргументирам".

Програмирането отдавна вече е достигнало качествено ново състояние – да се програмира пълноценно вече може единствено при съзнателно и задълбочено аргументиране. Дори е желателно, а често е направо и задължително, съжденията да бъдат доказвани със строгост, типична за математиката. В много случаи за това се привличат и цели математически теории. Такова положение на нещата се обяснява с изключителната сложност на софтуера и производството му, а и на методите за оценка на алгоритмичната трудност на възникващите задачи. Всъщност, за програмистите е много типичен начинът на мислене чрез обосноваване (явно или подразбиращо се), чрез постоянно формулиране и отговаряне на въпроси "Защо...?". Много често увереността в практическата осъществимост на дадена система се крепи главно върху разсъждения, строго доказващи някакви специфични за съответната система твърдения.

Всяко умение от избраните по-горе четири типа и, следователно, самото усвояване на фундаменталните знания за програмирането, се постига единствено при овладяването на специфичното за тази област мислене чрез аргументиране и доказване. Но то може да бъде различно в различните подобласти.

В програмистката дейност особено място заема аргументацията в **о**бектно-**о**риентираното **п**рограмиране (ООП), защото самото то е дълбоко обосновано на базата на целия натрупан досега опит от работата с компютри и защото начинът на мислене в процеса на ООП е неделимо преплетен с много от най-ценните концепции в производството и използването на софтуер. Например, идеите на структурното програмиране намират естествено и особено плодотворно продължение в ООП (в определен смисъл или в определено направление, тези идеи достигат в ООП прецизен логически завършек).

Обаче, както във всяко програмиране, така и в ООП се прилагат два типа аргументиращо мислене, които имат и различно значение за горните четири умения.

Първо, редица съждения се оказват еднозначно следствие с математическа строгост произтичащо от по-рано доказани твърдения и вече приети решения.

Например, за да може един и същ код (да речем: . . . Obj.Elm. . .) без претипизиране да работи с елементите на разнотипни обекти (Obj да може да бъде от различен динамичен клас), трябва техните класове (динамичните класове на Obj) да са в обща йерархия. Това следва от концепцията в ООП за разширена съвместимост на класовете (която пък е следствие от свои основания). Също така, трябва инстанцията (Obj), цитирана в обработващия код, да бъде декларирана чрез (да има формален тип) класа, от който започва йерархията, а съответният цитиран елемент (Elm) да бъде деклариран още в същия този клас в началото на йерархията. По-нататък, за да може кодът да предизвиква различни действия, когато са различни динамичните класове на обработвания обект (Obj), и това да става без проверка на типа на обекта по време на изпълнението на програмата, задължително трябва обработката пряко или косвено да активира виртуален или динамичен метод (това би могъл да бъде самият Elm), деклариран като такъв също в началото на йерархията.

Такива строги разсъждения с еднозначно следствие, които с математическа прецизност доказват някакъв извод, са особено полезни. От една страна, защото разкриват задължителните аспекти на съществуващите езици и системи и така пряко развиват уменията У.1, У.2 и У.3. От друга страна, понеже посочват условията, които непременно трябва да изпълнява разработваният софтуер, а това дава важни ориентири при създаването на необичаен или сложен софтуер, т. е. силно подпомага умението У.4.

Вторият тип аргументиращо мислене се основава на здравия програмистки усет, на добрия вкус. При него са възможни няколко различни следствия от разсъжденията и изборът на едно от тях е търсене на баланс между противоречащи си изисквания. Типичното тук е, че аргументирането проследява, предвижда какво следва от всеки допустим избор.

Ето някои примери за възможен избор: Обекти с различни, макар и еквивалентни, набори от полета могат да описват еднаква същност, но тези варианти на полета създават и различни удобства или неудобства, включително променят реализацията на някои методи. (Така триъгълникът може да се представи с три страни или със страна и два ъгъла и различно ще се изчислява площта му.) Изборът на частни полета може да наложи създаването на защитени методи за достъп до тях, а при избор на защитени полета отпада необходимостта от методите, но се намалява контролът върху полетата. Наличието на поле указател към динамична данна може да спести памет, но изисква специални деструктор и начин на копиране. Възможно е в един обект да има указател, насочван към разнотипни други обекти, но може да се използват и различни инстанции без указател, вграждащи в себе си съответните други обекти. (Така производителят може да се асоциира с произвеждана стока чрез указател към нея или описанието на стоката да бъде част от обекта производител.) Обаче, в двата случая следват принципно различни йерархия и гъвкавост на модела и съвсем различно се работи с обектите.

По същество, здравият програмистки усет е обобщение от многобройни относително доказани частни случаи, но такова, за което е неприемливо трудно да бъде строго доказано. Развиването на здрав вкус всъщност е придобиване на обобщено знание и на умение за неговото използване в нови ситуации.

Аргументирането при избор на една от няколко възможности много осезаемо и ефективно научава студентите на самостоятелно мислене и на откриване на опорни-

те твърдения, чрез които може надеждно да се анализира нова, нетрадиционна или твърде сложна задача или област. Такова аргументиране е много ценно за уменията У.1, У.2 и У.3, защото внася яснота и увереност там, където строгите детерминирани разсъждения са приложими само отчасти и са недостатъчни за търсенето на правилни решения.

Що се отнася до умението У.4, за него такова аргументиращо мислене с привличане на интуиция е задължително, защото тъкмо при необикновените и сложните задачи се построяват относително дълги редици от надграждащи се аргументи и многократно се налага избор при неоднозначни следствия. Когато в подобен процес се натрупат голям брой неудачни решения, това по правило води до практическа недостижимост на съответната поставена крайна цел.

Всъщност, в ООП аргументирането е значително по-строго, отколкото в програмирането без използване на класове, и много по-често се достига до еднозначни изводи. В ООП причинно-следствените зависимости са по-дълбоко преплетени във всички етапи и елементи на процеса на създаване на софтуерна система. При това, в колкото по-напреднала фаза се намира разработката, толкова по-еднозначни стават изводите от вече приетите решения, толкова по-тесен става кръгът на възможните варианти. Такова засилване на строгата предопределеност на изводите е типично и естествено за ООП, получава се сякаш само. В същото време, когато не се използват класове, аналогичното стесняване на възможностите за избор се опира повече на съзнателния човешки стремеж, на програмистката дисциплина, и твърде лесно би могло да се пренебрегне. Затова в ООП в съществено по-голяма степен е възможно, точно както в математиката се доказва теорема, крайният софтуерен продукт да бъде построен като следствие именно от аргументи, а не като отражение на нечии предпочитания, които биха могли да са откъснати от вътрешната логика и на задачата, и на използваните технологични средства.

Посочените по-горе примери за аргументи съществено подпомагат студентите и при разбирането на теоретичния материал, и при решаването на практически задачи. За ООП е типично, че подобни аргументи съществуват на всяко ниво от проектирането и програмирането. Ето и още няколко примера, които също са много полезни за обучаемите, защото дават опорни съждения, ориентири, които лесно, естествено и убедително посочват правилните решения в съответните ситуации (списъкът от примери в никакъв случай не е изчерпателен):

При избора на полета: Наборът от полета трябва еднозначно да описва представяната същност, защото това е естественото предназначение на обекта. Не бива никое поле да бъде функционално зависимо от останалите, понеже така се създава риск за грешки при модифициране и излишно се губи памет. Стойност, еднаква за всички обекти от класа, не се съхранява в поле на обектите, защото инстанцията трябва да "помни" само уникалните атрибути на представяната същност. (Така числото π не бива да бъде поле на кръга.)

При избора на методи: Само алгоритъм, използващ елементи от обекта, има смисъл (но не винаги е задължително) да бъде метод. Обработка, която променя полета на обект, пряко или косвено трябва да активира метод (или самата тя да бъде метод), за да се контролира състоянието на обекта.

При наследяване: Наследените методи, чието действие не е адекватно на производния клас, трябва да се предефинират, за да съдържа обектът само елементи,

съответстващи на представяната същност. В частност, ако метод от базовия клас изписва при диалог типа на обекта, в производният клас трябва да се предефинира наследеният метод, вместо да се създава нов метод с нов диалог.

При използването на полиморфизъм: За да може даден базов клас да се наследява по всякакви начини, в него трябва да има виртуален деструктор. Това позволява чрез указател, деклариран с базовия клас, да се унищожават правилно обекти и от производни класове, които изискват заключителна обработка, различна от предвидената в базовия клас. За да може една подпрограма да работи различно, когато се извиква с разнотипни обекти, трябва нейният формален параметър да дава достъп до фактическия параметър, т. е., параметрите да се свързват по адрес, указател, псевдоним.

Възможни са и много други примери.

Надграждането на подобни аргументи от началния до заключителния стадий от разработката на дадена система превръща нейното проектиране и програмиране в процес на доказване и на записване на получаваните изводи с формализиран език. ООП предразполага към такъв начин на работа повече и със сигурност дава възможности за това съществено повече, отколкото програмирането без използване на класове.

4. Въздействието на аргументиращото мислене в ООП върху мотивацията на учащите и чрез нея – върху фундаменталната подготовка. От гледна точка на ролята на аргументиращото мислене специално внимание заслужава мотивацията на учащите, тъй като тя е един от най-важните фактори за успешното обучение въобще, и за горните четири умения, в частност. Аргументиращият начин на мислене не само обучава студентите и развива способностите им. Процесът на аргументиране разкрива пред тях най-интересните, най-увлекателните аспекти на създаването на софтуер. Проследяването на зависимостите и причинно-следствените връзки позволява на учащите пределно ясно да почувстват най-красивите страни на програмирането, точно онези, които винаги, у кого по-малко, а у кого повече, завладяват въображението, очароват и предизвикват желание за работа.

Освен това, аргументиращият начин на мислене, особено в ООП, е мощно средство не само в програмистката дейност, но и в цялостното обучение. Овладявайки непосредствения инструмент, чрез който могат да преодоляват срещаните трудности и да реализират действащи системи, студентите придобиват увереност в собствените сили, започват да чувстват, че наистина са в състояние да осъществяват най-привлекателните за тях замисли.

Интересът и желанието за работа нарастват още повече, когато обучаемите се убеждават, че днес обекти и класове се използват буквално навсякъде в компютърната информатика, т. е., че чрез ООП може да се създава всякакъв софтуер.

Така аргументиращото мислене в ООП се превръща в стимулатор и източник на дълбока мотивация, която пък от своя страна много силно влияе на изброените по-горе четири умения. В частност, показателно е, че най-нетрадиционните и най-сложните задачи, за които се отнася умението У.4, а също и най-интересните, обикновено се решават от изключително мотивирани личности.

5. Изводи. От всичко казано следва, че за изграждането на фундаменталните знания по програмиране необходимо условие е развиването на задълбочено аргумен-

тиращо мислене в ООП. А един от главните пътища за постигането на този ефект е силното стимулиращо и мотивиращо въздействие, което оказва такова мислене върху студентите.

Очевидно е обаче, че това необходимо условие не е и достатъчно за построяването на желания фундамент. Защото например, редица основополагащи идеи, свързани с компютърни мрежи, компютърна графика, бази от данни, изкуствен интелект и др., се разпростират и далеч извън областта на ООП.

Кирил Иванов
катедра Компютърна информатика
Факултет по Математика и Информатика
Пловдивски Университет "Паисий Хилендарски"
бул. България № 236
4003 Пловдив
e-mail: kiv@uni-plovdiv.bg

PROOF-ORIENTED THINKING IN THE OBJECT-ORIENTED PROGRAMMING WHEN BUILDING FUNDAMENTAL PROGRAMMING KNOWLEDGE

Kiril Ivanov

Four criteria for estimating the degree of fundamental programming knowledge acquisition are formulated. The specificity of the proof-oriented thinking in object-oriented programming and its role in the learning of fundamentals are pointed. Two ways of reasoning are distinguished: with an only possible conclusion and with a multiple choice by search of balance between contradictory requirements. Examples of arguments that help considerably the students to understand the basic ideas related to the use of objects and classes in different stages of the software system development are given. Particular attention is paid to the influence of the proof-oriented thinking on the learners' motivation and hence – on their fundamental knowledge acquisition.