UNELE ASPECTE METODOLOGICE ALE PREDĂRII LIMBAJELOR DE PROGRAMARE

Mircea PETIC, *dr., lector sup. univ.,*
Universitatea de Stat „Alecu Russo" din Bălți, Republica Moldova

Summary: *The article presents some ideas that can constitute the foundations for developing a methodology for teaching different programming languages both at the university and in high schools. It treats aspects of presentation of material related to programming languages, as well as possible skills improvement and self-improvement of students who wish to study programming languages. Issues related to the presentation of software development environment tools to write a good program code have been discussed as well as how to test students' written programs.*

Key-words: *methodology, teaching, programming language, software development environment.*

Cercetătorii Hayes şi Bloom afirmă că durează aproape zece ani pentru a căpăta competenţă în oricare dintr-o largă varietate de domenii fără excepţie, incluzînd şi programarea. Una din sarcinile de bază ale studentului de la specialităţile legate de programare este studierea limbajelor de programare. Importanţa studierii limbajului de programare reiese imediat din principala funcţie a lui, cea de mijloc de interacţiune practică dintre om şi calculator. În acest sens, este important de a alege metodologia corectă de predare a limbajelor de programare (Masalagiu&Asiminoaiei, 2004).

Cea mai firească metodologie de predare a unui limbaj de programare este utilizarea multiplelor exemple concrete de rezolvare a unor probleme, urmînd a se intra ulterior în detalii privind elementele de limbaj utilizate în diversele implementări. Totuşi această metodă poate fi utilizată mai ales în cazul unor studenţi neiniţiaţi în tehnologia programării, deoarece ar putea apărea plictiseala din cauza teoretizării excesive.

O altă abordare constă în orientarea pe instrucţiuni în cazul predării unui limbaj de programare. În acest caz metoda este adoptată de către cei care consideră că limbajul este o mulţime de instrucţiuni şi care predau aceste elemente într-o ordine oarecare, stabilită pe baza unor considerente particulare. În acest sens este vorba de instrucţiunile alternative (if, case/switch), structurile repetitive (for, foreach, while, repeat/dowhile, loop).

Dacă se consideră că importanţa majoră o deţine conceptul de bază al limbajului de programare atunci se va preda după acest concept, apoi se va dezvolta prezentarea pe baza conceptelor derivate. Această orientare seamănă cu cea orientată pe probleme, dar în continuu ţine cont de puncte de vedere generale. Elementele de limbaj sau de mediu se vor introduce pe parcurs, în funcţie de cerinţele dirijate de conceptul prezentat (Maxim&Moroşan, 2007).

Din alt punct de vedere un limbaj de programare poate fi privit ca ceva indivizibil în care primează logica, filosolia lui şi elementele sale sînt introduse pe parcursul prezentării pe baza acestei filosofii. O asemenea abordare contrazice regula de bază în predarea informaticii, şi anume: se porneşte de la probleme concrete. Dar în cazul unui aşa limbaj de programare Java este important să fie introdus conceptul de Maşină Virtuală Java. În acest sens este important să se pornească de la faptul că una dintre cerinţele care au stat la baza creării limbajului Java a fost aceea de a obţine un limbaj portabil. Aceasta ar însemna că un program scris pe un anumit calculator, pe care rulează un anumit sistem de operare, să poată rula fără modificări şi pe un alt calculator, pe care rulează alt sistem de operare. De exemplu, să se poată scrie un program care să fie rulat pe Windows şi pe Linux.

Indiferent de abordarea concretă, cel mai important este în programare să programezi. În acest sens materialul teoretic trece pe planul doi, dacă nu se face accentul pe lucrul practic. Contează comunicarea între studenţi de orice fel: citirea programelor unul altuia, analizarea şi discutarea lor. În acest sens proiectele de programare în grup fiind unul din cele mai eficiente metode de învăţare. Deoarece fiind cel mai bun într-un grup îi poţi învăţa pe alţii şi dacă eşti mai slab poţi învăţa de la cei mai buni. Numărul membrilor unui grup poate varia de la 2 la 10, dar cele mai potrivite grupuri sînt cele formate din 4-6 membri. O altă particularitate este ca în general să fie sarcini bine definite pentru fiecare membru a grupului. Totuşi se impută pe bună dreptate acestei munci în grup o eficienţă scăzută. Diversificarea sarcinilor grupurilor şi împărţirea sarcinilor între membrii grupurilor atenuează această deficienţă.

Dacă prin activitatea de grup se intenționează dobîndirea de noi cunoștințe prin lucrul cu manualul, documentația sau prin testarea unor produse soft, pentru profesor este obligatoriu de a organiza dezbateri finale, care să stabilească dacă membrii grupurilor și-au însușit corect noțiunile și și-au format deprinderi corecte. Este la fel interesant de a incita și de a coordona discuțiile între grupuri în direcția obținerii concluziilor care se impun. Astfel cînd unul sau mai multe grupuri găsește o soluție, acestea vor fi discutate și analizate succesiv sau în paralel. Scopul acestei discuții este de a reliefa corectitudinea rezolvării, determinarea celei mai eficiente și mai elegante soluții și de a descoperi eventualele erori (Maxim&Moroșan, 2007).

Un alt aspect important este particularitatea limbajului de programare ales pentru îndeplinirea sarcinilor concrete date de o problemă complexă. În acest sens este important de a ține cont de acest lucru, deoarece fiece limbaj de programare are menirea pentru a rezolva anumite probleme. Așa cum numărul limbajelor de programare este mare, o clasificare a lor ar fi utilă. O încercare de clasificare a limbajelor în baza unor criterii:

1. După nivelul de apropiere al acestora de limbajul natural
   - Limbaje de nivel scăzut: limbajul de asamblare
   - Limbaje de nivel mediu: C, C++
   - Limbaje de nivel înalt: Java, PHP, Prolog, Pascal, Python, etc.
2. După modul de „traducere”
   - Limbaje compilate: C, C++, Pascal;
   - Limbaje interpretate: PHP, Prolog, Matlab
3. După scopul pentru care au fost proiectate
   - Limbaje de uz general: C, C++, Pascal
   - Limbaje specializate: Prolog, SQL.
4. După modul de restricționare
   - Limbaje tipizate: Java, C, C++, Pascal
   - Limbaje netipizate: PHP

Iată de ce este necesar în cadrul instruirii de a selecta anume sarcinile care ar evidenția caracteristicile specifice ale unui anumit limbaj de programare.

Referindu-ne la predarea limbajelor de programare nu trebuie să uităm de mediul de dezvoltare specific pentru că cunoașterea posibilităților lui asigură celui care învață limbajul de programare concret la unelte concrete la scrierea unui cod de program bun. Prin mediu de dezvoltare (engl. software development enviroment, sau integrated development enviroment) este un set de programe care ajută programatorul în scrierea programelor. Un mediu de dezvoltare combină toți pașii necesari creării unui program într-un singur soft, care, de regulă, oferă o interfață cu utilizatorul grafică și prietenoasă. Principalele componente ale unui mediu de dezvoltare sînt editorul de cod sursă și depanatorul. Mediile de dezvoltare apelează compilatoarele sau interpretoare, care pot veni în același pachet cu mediul însuși, sau pot fi instalate separat de către programator. Printre facilitățile prezente în mediile de dezvoltare mai sofisticate se numără: exploratoare de cod sursă, sisteme de control al versiunilor, designere de interfețe grafice, sau unelte de ingineria programării (ex. generarea de diagrame UML). De obicei un mediu de dezvoltare este specific unui anumit limbaj de programare, însă există la ora actuală și medii de dezvoltare care pot lucra cu mai multe limbaje, de ex. Eclipse sau Microsoft Visual Studio. Acest aspect este

important prin prisma procesului de interoperabilitatea limbajelor de programare şi etapă importante în obţinerea de aplicaţii de calitate (Camerzan&Vascan, 2010).

Nu trebuie să fie dat uitării nici specificul de testare a programelor, care presupune eliminarea erorilor. Modul lor de eliminare poate fi realizat în mod diferit. Astfel pentru a vedea dacă programul funcţionează corect acesta trebuie executat cu anumite seturi de date numite teste. În acest sens dacă a fost dată pe acasă sau chiar şi în timpul orelor o problemă pentru a fi programată, pentru a se convinge profesorul de modul corect de rezolvare el recurge la alcătuirea unor teste concrete. Iată de ce aşa cum profesorul fizic nu reuşeşte sa verifice concomitent la toţi studenţi programele scrise, şi în aceste condiţii poate recurge la serviciu de poştă electronică pentru a transmite fişierele sursă a programelor ca în afara orelor de curs să fie verificate, aşa cum în timpul orelor verificarea durează mult. În plus trimiterea programelor prin email sau alte utilitare în timp real precum Yahoo Messenger sau Skype permite chiar economisirea timpului şi oferirea unui răspuns rapid în cazul unei probleme apărute la scrierea codului programului pentru a fi prezentat la ore de laborator. Dincolo de asta obţinerea fişierelor sursă de cod a programelor care trebuie scrise oferă profesorului o posibilitate de orientare personală la eventuala analiză la lecţie şi elaborarea sarcinilor ulterioare pentru corectarea greşelilor [4].

Totuşi această abordare de verificare programelor poate fi îmbunătăţită. În acest sens folosirea evaluatorilor automaţi este utilă, deoarece permite verificarea în timp rapid a mai multor aplicaţii, folosind mai multe mulţimi de teste. Evaluatorii automaţi reprezintă programe externe pentru verificarea răspunsurilor. În cazul programelor scrise care trebuie verificate evaluatorul le prelucrează în modul următor. Textul iniţial este compilat în modul stabilit. Dacă în cazul compilării apar erori de compilare, verificarea soluţiei se termină la această etapă cu mesajul de eroare la compilare („Compilation error”). De menţionat că în cazul evaluatorilor timpul de compilare este limitat. Dacă compilarea programului a necesitat mai mult timp, decît timpul stabilit pentru îndeplinire, la fel va fi semnalată o stare de eroare de compilare. În cazul în care compilarea programului s-a terminat cu succes, fişierul executabil este transmis la testare. Există un număr de teste pregătite din timp. Pot fi setate la evaluator mai multe variante: să fie testat programul pentru toate testele, pentru o parte din ele, sau doar pînă la testul care nu a fost corect.

În afară de faptul dacă un test a fost trecut sau nu, mai sînt şi alte rezultate de execuţie a programului care la prima vedere, doar parcurgînd cu ochiul şi văzînd codul programului este imposibil de observat. Astfel evaluatorul permite de a stabili:

- Erorile de compilare
- Erori de executare
- Depăşirea timpului de execuţie
- Formatul greşit a datelor rezultat
- Răspuns greşit
- Eroarea sistemului de testare
- Depăşirea limitei de memorie

Folosirea evaluatorului nu doar economiseşte timpul pentru profesor şi student pentru a stabili erorile în execuţia programului, dar şi disciplinează studentul pentru ca să se înveţe să realizeze programele în strictă legitate cu normele stabilite de problema concretă. În acest sens programul nu poate evalua greşit programul scris şi

dat spre testare. Evaluatorii pot fi scrişi şi găsiţi pentru diferite limbaje de programare, de la Pascal şi C/C++ pînă la Java. Aceasta permite chiar lucrul individual în afara orelor de curs şi fără a intervenţia profesorului.

În concluzie, procesul de predare a limbajelor de programare este unul interesant care merită să fie studiat. Predarea nu poate fi indivizibilă cu procesul permanent de învăţare de sine stătătoare, de lucrul individual, iată de ce sînt importante sarcinile continue care ar incita studentul de a munci şi de a-şi perfecţiona capacităţile de programare. Ţinînd cont de particularităţile concrete a limbajului de programare care este predat trebuie aleasă o metodologie bine gîndită, care poate fi pusă în practică şi care ar putea avea succes. O abordare adecvată situaţiei concrete procesului de predare a unui anumit limbaj contribuie la formarea competenţelor practice în găsirea soluţiei potrivite în procesul de programare.

Referinţe bibliografice

1. C. Masalagiu, I. Asiminoaei. Didactica predării informaticii, Editura Polirom, 2004.
2. Maxim, C. Moroşan, Informatică. Didactica specialităţii. Iaşi, 2007
3. Camerzan, T. Vascan, Didactica informaticii: Suport didactic, Editura „Elena-V.I", Chişinău, 2010, 200 p.
4. M. Petic. E-learning în şcoală: domenii şi căi de realizare. Experienţa LT „Ion Creangă", Bălţi. În: Materialele Conferinţei Ştiinţifico-Practice „Promovarea tehnologiilor informaţionale şi comunicaţionale în educaţie. 26-27 iunie 2009, Chişinău, 2009, pp. 50-58.