Η ΕΠΙΛΟΓΗ ΤΗΣ ΚΑΤΑΛΛΗΛΗΣ ΕΙΣΑΓΩΓΙΚΗΣ ΓΛΩΣΣΑΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΓΙΑ ΑΡΧΑΡΙΟΥΣ

Εφόπουλος Βασίλειος
Τμ. Εφαρμοσμένης Πληροφορικής
Παν. Μακεδονίας
efop@uom.gr

Δαγδιλέλης Βασίλειος
Τμ. Εκπαιδευτικής και Κοινωνικής
Πολιτικής, Παν. Μακεδονίας
dagdil@uom.gr

Ευαγγελίδης Γεώργιος
Τμ. Εφαρμοσμένης Πληροφορικής, Παν. Μακεδονίας
gevan@uom.gr

ΠΕΡΙΛΗΨΗ
Η παρούσα εισήγηση περιγράφει τα αποτελέσματα ερευνών που έχουν γίνει με αντικείμενο την επιλογή μιας γλώσσας κατάλληλης για τη διδασκαλία και εκμάθηση του προγραμματισμού για αρχαρίους.

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: Προγραμματισμός**,** Γλώσσα Προγραμματισμού, Αρχάριοι προγραμματιστές

ΕΙΣΑΓΩΓΗ

Από τις σύγχρονες επιστήμες, οι λεγόμενες *θετικές επιστήμες* είναι αναμφίβολα αυτές που έχουν τη μερίδα του λέοντος στο εκπαιδευτικό λογισμικό αλλά και στις εκπαιδευτικές χρήσεις του λογισμικού γενικής φύσεως – όπως είναι η περίπτωση των λογιστικών φύλλων που χρησιμοποιούνται στη διδασκαλία των Μαθηματικών ή των επεξεργαστών κειμένου που χρησιμοποιούνται στη γλωσσική διδασκαλία.

Στην κατηγορία του λογισμικού για θετικές επιστήμες, η Πληροφορική αριθμεί πολλές δεκάδες εκπαιδευτικών περιβαλλόντων που χρησιμοποιούνται για τη διδασκαλία της. Η μεγάλη πλειοψηφία των περιβαλλόντων αυτών είναι προσανατολισμένη κυρίως στην παρουσίαση της τεχνολογίας των πληροφορικών συστημάτων – όπως τα δίκτυα, το δυαδικό σύστημα και οι επεξεργαστές - και τον προγραμματισμό. Τα περιβάλλοντα για τη διδασκαλία του προγραμματισμού μάλιστα είναι ίσως τα πιο πολυάριθμα από όλες τις σχετικές κατηγορίες εκπαιδευτικού λογισμικού, γεγονός που εξηγείται από την κοινά αποδεκτή σήμερα διαπίστωση ότι ο προγραμματισμός παρουσιάζει πολλές δυσκολίες κυρίως για τους αρχάριους προγραμματιστές.

Τα περιβάλλοντα αυτά που χρησιμοποιούνται για τη διδασκαλία του προγραμματισμού δεν στηρίζονται όλα στις ίδιες διδακτικές αρχές, ούτε φυσικά στην ίδια τεχνολογία και παρουσιάζουν μια πολύ μεγάλη ποικιλία στη διεπαφή τους. Με ένα γενικό τρόπο ωστόσο, τα περιβάλλοντα αυτά παρουσιάζουν ορισμένα κοινά χαρακτηριστικά, όπως για παράδειγμα το γεγονός ότι κατά κανόνα πρόκειται για περιβάλλοντα τα οποία στηρίζουν κάποιου είδους γλώσσα προγραμματισμού. Σε μερικές μάλιστα περιπτώσεις, η γλώσσα και το περιβάλλον προγραμματισμού αποτελούν ένα ενιαίο σύνολο στο οποίο δύσκολα μπορεί κανείς να διακρίνει την καθεμιά από τις οντότητες «περιβάλλον» και «γλώσσα προγραμματισμού», όπως για παράδειγμα στο AgentSheets (Repenning, 1995).

Οι γλώσσες αυτές μπορούν να διαιρεθούν σε δυο κατηγορίες:

i. γλώσσες που εντάσσονται στο εκάστοτε ισχύον προγραμματιστικό μοντέλο. Στη δεκαετία του 1970, η Pascal (Wirth & Jensen, 1975), για πολλά χρόνια θεωρείτο η κατάλληλη γλώσσα για την εισαγωγή στον προγραμματισμό, την εποχή της κυριαρχίας του μοντέλου του δομημένου προγραμματισμού όπως και μετέπειτα η Turing (Holt & Hume, 1984), η Modula-2 (Wirth, 1988), και λίγο αργότερα η Thetis C (Freund & Roberts, 1996) στην ίδια ακριβώς λογική. Την τελευταία δεκαετία, γλώσσες και περιβάλλοντα όπως η Blue (Kölling & Rosenberg, 1996) και η μετεξέλιξη της, BlueJ, είναι σύμφωνες με το μοντέλο του αντικειμενοστραφούς προγραμματισμού. Στις περιπτώσεις αυτές πολύ συχνά οι χρησιμοποιούμενες γλώσσες αποτελούν ελάσσονες εκδόσεις των τυπικών γλωσσών προγραμματισμού, που έχουν ως σκοπό τη μείωση του όγκου των γνώσεων που πρέπει να αποκτήσει ο αρχάριος προγραμματιστής. Πρόκειται για τις λεγόμενες «μικρο-γλώσσες» (Brusilovsky et al, 1997).

ii. γλώσσες οι οποίες δεν εντάσσονται στο κλασικό πρότυπο του προγραμματιστικού «παραδείγματος» και αποσκοπούν αποκλειστικά στη διευκόλυνση της εκμάθησης μερικών βασικών εννοιών της αλγοριθμικής και του προγραμματισμού. Ο κύριος εκπρόσωπος της κατηγορίας αυτής είναι βέβαια η Logo (Papert, 1980) και το ρομπότ Karel (Pattis, 1981), αλλά και η StarLogo, τα ρομπότ Legologo και πιο πρόσφατα τα Mindstorms. Στην ίδια κατηγορία εντάσσονται επίσης και περιβάλλοντα-γλώσσες όπως τα AgentSheets (Repenning, 1995) και ToonTalk – απλώς για να αναφέρουμε μερικά γνωστά παραδείγματα από τα πολυάριθμα που υφίστανται.

Και στις δυο κατηγορίες γλωσσών και περιβαλλόντων που αναφέρονται παραπάνω, ο βασικός σκοπός, που είναι η κατανόηση και η εκμάθηση της αλγοριθμικής και του προγραμματισμού, επιβάλλει την ύπαρξη ορισμένων πρόσθετων χαρακτηριστικών στη διεπαφή (για παράδειγμα ο προγραμματισμός μπορεί να στηρίζεται σε ειδικούς εκδότες), στη ρύθμιση της ταχύτητας ροής του προγράμματος, στην προσομοίωση της εκτέλεσης του κώδικα σε διάφορα επίπεδα, στην οπτικοποίηση ορισμένων λειτουργιών της διαδικασίας εκτέλεσης.

Η συστηματική μελέτη των γλωσσών και περιβαλλόντων αυτών, αποκαλύπτει ωστόσο δυο σημαντικά στοιχεία:

i. τον υπερβολικά μεγάλο αριθμό τους και την σχετικά περιορισμένη, στην πλειοψηφία των περιπτώσεων, επίδραση τους στην εκπαιδευτική κοινότητα. Ένας πολύ μεγάλος αριθμός των γλωσσών-περιβαλλόντων αυτών ουσιαστικά χρησιμοποιούνται «τοπικά», δηλαδή σε έναν σχετικά περιορισμένο χώρο και χρόνο και γίνονται ευρύτερα γνωστές κυρίως μέσα από την ακαδημαϊκή δραστηριοποίηση (δημοσιεύσεις) ατόμων και ομάδων που σχετίζονται με την επινόηση και κατασκευή τους.

ii. την υπερβολική ομοιότητα και επανάληψη χαρακτηριστικών. Όχι μόνο τα πολυάριθμα περιβάλλοντα Logo-like είναι όμοια, αλλά και σε πολλές άλλες περιπτώσεις, χαρακτηριστικά οπτικοποίησης ή και της διεπαφής επαναλαμβάνονται σχεδόν στερεότυπα σε κάθε νέο περιβάλλον που δημιουργείται – χωρίς να είναι πάντοτε βέβαια η διδακτική τους αποτελεσματικότητα.

Θα ήταν αναμενόμενο μέσα στο πλαίσιο αυτό, αν μάλιστα λάβει κανείς υπόψη του τις παραπάνω παρατηρήσεις που σχετικοποιούν σε κάποιο μέτρο τη διδακτική αποτελεσματικότητα των διδακτικών περιβαλλόντων, ότι οι γλώσσες αυτές πρέπει να ενσωματώνουν τα πρόσφατα ευρήματα και πορίσματα ερευνών που σχετίζονται με τις δυσκολίες που συναντούν οι αρχάριοι προγραμματιστές. Ωστόσο, παρόλο τον παιδαγωγικό τους στόχο, φαίνεται ότι οι γλώσσες αυτού του είδους δε σχεδιάζονται κατά κανόνα με γνώμονα τις πρόσφατες παιδαγωγικές παραδοχές (Pane & Myers, 2000). Στην πραγματικότητα η σχεδίαση μιας γλώσσας προγραμματισμού, όπως

δείχνει η σχετική έρευνα, σπάνια λαμβάνει υπόψη τις βασικές αρχές σχεδίασης του περιβάλλοντος διεπαφής και τη γνώση μας για τις δυσκολίες που συναντούν οι αρχάριοι στην εκμάθηση του προγραμματισμού.

ΚΡΙΤΗΡΙΑ ΕΠΙΛΟΓΗΣ ΓΛΩΣΣΑΣ ΓΙΑ ΔΙΔΑΣΚΑΛΙΑ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Η επιλογή μιας γλώσσας προγραμματισμού που θα είναι κατάλληλη για τη διδασκαλία και την εκμάθηση του προγραμματισμού σε αρχαρίους είναι ένας σημαντικός παράγοντας για την επιτυχία της όλης διαδικασίας. Είναι γεγονός ότι υπάρχει μια τάση, που ορισμένες φορές φτάνει τα όρια της πίεσης, για χρήση εμπορικών γλωσσών προγραμματισμού. Αρκετοί φορείς που έχουν ακολουθήσει αυτή την τάση και χρησιμοποιούν για την εισαγωγή στον προγραμματισμό εμπορικές γλώσσες όπως λ.χ. η C++, η Java, εξηγούν την απόφαση τους με το δεδομένο ότι οι μαθητές ανταποκρίνονται θετικά στην ιδέα να μάθουν μια γλώσσα που θα μπορούν να τη χρησιμοποιήσουν και μετά στον επαγγελματικό στίβο (Allen, 1996). Ο χρόνος που θα δαπανήσουν για να μάθουν μια εκπαιδευτική - και όχι εμπορική – γλώσσα, που δεν θα είναι άμεσα χρήσιμη έξω από την τάξη, μπορεί να τους απογοητεύσει και να θεωρήσουν το χρόνο τους χαμένο. Αυτό συμβαίνει ακόμα και αν αντιληφθούν ότι με αυτή τη γλώσσα διδάσκονται καλύτερα τις αρχές του προγραμματισμού (Conway, 1993). Ως αποτέλεσμα όλων αυτών των αντιλήψεων προκύπτει ότι η δραστηριοποίηση και το ενδιαφέρον των μαθητών στα μαθήματα που γίνεται χρήση εκπαιδευτικής γλώσσας δεν θα είναι στο ίδιο επίπεδο με τα αντίστοιχα μαθήματα με χρήση εμπορικής γλώσσας.

Είναι γεγονός ότι οι έρευνες στο παραπάνω ζήτημα είναι επικεντρωμένες κυρίως στο χώρο της τριτοβάθμιας εκπαίδευσης και σε φοιτητές που μαθαίνουν προγραμματισμό για να τον χρησιμοποιήσουν και στην επαγγελματική τους καριέρα. Στην περίπτωση όμως της δευτεροβάθμιας εκπαίδευσης, ο προγραμματισμός είναι γενικό μάθημα και η διδασκαλία του έχει διαφορετικούς στόχους. Από τη δική μας έρευνα προέκυψε ότι υπάρχουν δύο τάσεις στους μαθητές της δευτεροβάθμιας. Η μια τάση επιθυμεί τη χρήση εμπορικών γλωσσών και απαρτίζεται από μαθητές που είχαν κάποια σχετική επαφή με τον προγραμματισμό στο παρελθόν. Η άλλη τάση περιλαμβάνει μαθητές που δεν είχαν την παραμικρή επαφή και συνήθως θεωρούν την εκμάθηση του προγραμματισμού μια δύσκολη διαδικασία. Στη δεύτερη περίπτωση οι μαθητές προφανώς προτιμούν μια απλή εκπαιδευτική γλώσσα και ένα φιλικό περιβάλλον.

Η χρηστικότητα της πρώτης γλώσσας προγραμματισμού που μαθαίνει ένας μαθητής μπορεί να επιδράσει στη δραστηριοποίηση του και στον ενθουσιασμό του με πολλούς τρόπους. Εάν η γλώσσα δεν είναι ξεκάθαρη τότε η διαισθητική «ερμηνεία» του συντακτικού της γλώσσας οδηγεί σε εσφαλμένα αποτελέσματα. Ως συνέπεια οι μαθητές δεν εμπιστεύονται τη διαίσθηση τους, κάτι που καθιστά τη διαδικασία εκμάθησης του προγραμματισμού πιο δύσκολη και λιγότερο ευχάριστη. Εάν η γλώσσα είναι δύσκολη στην ανάγνωση και στη μετάφραση, αυτό επίσης επιδρά στη δραστηριοποίηση και οδηγεί τους μαθητές, ιδιαίτερα αυτούς που δεν έχουν αυτοπεποίθηση, να πιστέψουν ότι δεν έχουν το απαιτούμενο υπόβαθρο για να τα καταφέρουν στον προγραμματισμό.

Σύμφωνα με έρευνα που διεξήγαγε η ομάδα του Levy (1995) και αναφέρεται στην τριτοβάθμια εκπαίδευση, το 48,7% των ινστιτούτων που συμμετείχαν στην έρευνα, χρησιμοποιούν διαδικαστική γλώσσα στα μαθήματα εισαγωγής στον προγραμματισμό, το 36,2% χρησιμοποιούν αντικειμενοστραφείς γλώσσες, 12% χρησιμοποιούν Scheme, 2,4% χρησιμοποιούν άλλες functional γλώσσες όπως η ML και 0,7% επιλέγουν άλλες γλώσσες. Ακόμα σύμφωνα με τα αποτελέσματα της έρευνας που διεξήγαγαν οι Deek & Kimmel (1999) σε σχολεία δευτεροβάθμιας εκπαίδευσης στο New Jersey των Ηνωμένων Πολιτειών, στο 86% των σχολείων διδάσκεται η

γλώσσα Pascal, στο 71% διδάσκεται η γλώσσα BASIC, ακολουθεί η C με 16% ενώ στο 14% των σχολείων διδάσκονται άλλες γλώσσες. Στο 58% των σχολείων διδάσκονται και η Pascal και η BASIC.

ΠΑΙΔΑΓΩΓΙΚΑ ΠΡΟΒΛΗΜΑΤΑ ΠΟΥ ΣΥΝΑΝΤΩΝΤΑΙ ΣΤΙΣ ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

Αρκετές γλώσσες, που χρησιμοποιούνται ως εισαγωγικές στη διδασκαλία του προγραμματισμού, παρουσιάζουν σοβαρά παιδαγωγικά προβλήματα. Πολλές φορές οι σχεδιαστές τέτοιων γλωσσών θεωρούν ότι η αρχή «το λιγότερο είναι και περισσότερο» είναι αρκετή για να παρέχει λύση στο πρόβλημα. Αποδεικνύεται όμως ότι τέτοιες προσεγγίσεις δε βοηθούν τη διαδικασία μάθησης, ορισμένες μάλιστα φορές μπορούν να χαρακτηριστούν και καταστρεπτικές. Ένα χαρακτηριστικό παράδειγμα της παραπάνω προσέγγισης είναι η γλώσσα Scheme καθώς και όλες οι γλώσσες που στηρίζονται στη LISP. Η Scheme έχει ένα μόνο τύπο δεδομένων, τη λίστα, και μία μόνο λειτουργία, τον υπολογισμό σε μια λίστα. Αυτά τα χαρακτηριστικά καθιστούν ιδιαίτερα απλή την πρώτη επαφή των αρχάριων με τη γλώσσα. Το αποτέλεσμα όμως του κώδικα που παράγεται είναι πολύ δύσκολο να διαβαστεί και να κατανοηθεί επειδή περιέχει ένα μεγάλο αριθμό από εμφωλευμένες παρενθέσεις χωρίς να υπάρχει δομημένη παρουσίαση τους.

Από την άλλη πλευρά υπάρχουν πολλές γλώσσες προγραμματισμού που σχεδιάστηκαν με τελείως διαφορετική φιλοσοφία. Είναι οι γλώσσες που προσφέρουν όσο το δυνατό περισσότερα χαρακτηριστικά και δυνατότητες (όπως για παράδειγμα ή C, ή C++, η Ada). Από τις γλώσσες αυτές συνήθως μόνο ένα μικρό τμήμα τους διδάσκεται στους αρχάριους, αγνοώντας τα δυσνόητα και ισχυρά τους χαρακτηριστικά. Η διαδικασία αυτή όμως εμπεριέχει τα ακόλουθα προβλήματα:

- Το συνοδευτικό υποστηρικτικό υλικό της γλώσσας (εγχειρίδια χρήσης, βιβλία, βοήθεια on-line) δεν είναι προσαρμοσμένο σε αρχάριους χρήστες
- Ο μεταγλωττιστής της γλώσσας δεν περιορίζεται στο συγκεκριμένο υποσύνολο αλλά παράγει μηνύματα που αναφέρονται στο σύνολο της γλώσσας. Παράδειγμα τέτοιου μηνύματος λάθους είναι η χρήση ως ονόματος μιας δεσμευμένης λέξης της γλώσσας που δε τη γνωρίζει ο αρχάριος. Πολλά δε από τα μηνύματα λάθους του μεταγλωττιστή δεν είναι κατανοητά από τον αρχάριο χρήστη.

Πολλές από τις γλώσσες αυτές ενσωματώνονται σε ολοκληρωμένα εμπορικά περιβάλλοντα, αποτελώντας συνολικά μια "εμπορική λύση" που παρουσιάζει και αυτή με τη σειρά της αρκετά προβλήματα.

Η ΛΥΣΗ ΜΙΑΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΟΥΣ ΓΛΩΣΣΑΣ

Οι βασικές προγραμματιστικές δεξιότητες είναι ουσιώδεις για την επιτυχία σε οποιοδήποτε προγραμματιστικό πρότυπο. Όπως όλο και περισσότερο γίνεται παραδεκτό, ειδικά για τη σχεδίαση μεγάλων συστημάτων ο αντικειμενοστραφής (object-oriented) προγραμματισμός είναι μια αποτελεσματική τεχνολογία που παράγει συμπαγή και επαναχρησιμοποιήσιμο κώδικα.

Όμως αναπτύσσοντας ένα αντικειμενοστραφές σύστημα απαιτείται για τις μεθόδους συγγραφή κώδικα και αυτό εμπλέκει διαχείριση δομών δεδομένων και έλεγχο ροής, που είναι τα κομβικά στοιχεία ενός παραδοσιακού μαθήματος στον προγραμματισμό. Πρόσφατες μελέτες (Duke. et. al, 2000) αναφέρουν ότι οι νέοι χρήστες που έρχονται σε επαφή με τον προγραμματισμό με ένα αντικειμενοστραφές περιβάλλον μπορεί να αποτύχουν να αποκτήσουν τις βασικές προγραμματιστικές δεξιότητες. Όπως πιο συγκεκριμένα αναφέρεται:

"στα επόμενα χρόνια, οι μαθητές που δεν έχουν κατάλληλα εξοικειωθεί με βασικές προγραμματιστικές δεξιότητες (όπως η χρήση βρόχων while και λογικών εκφράσεων για να

ανακαλύψουν την εσωτερική λογική του συστήματος) μπορεί να είναι ικανοί να δημιουργήσουν σχεδιασμούς σε υψηλότερο επίπεδο αλλά μοχθούν ιδιαίτερα να μετατρέψουν αυτούς τους σχεδιασμούς σε πραγματικό κώδικα".

ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΑΦΗΣ ΚΑΙ ΔΙΑΔΙΚΑΣΤΙΚΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Ένας σημαντικός αριθμός από πρόσφατες έρευνες αναφέρονται στον αντικειμενοστραφή (object–oriented) προγραμματισμό (λ.χ. με χρήση C++ ή Java), ειδικότερα σε σχέση με τον διαδικαστικό (procedural) προγραμματισμό (λ.χ. με χρήση C ή Pascal). Παρόλο που οι έρευνες αυτές αναφέρονται στην τριτοβάθμια εκπαίδευση, δίνουν μια αίσθηση και δηλώνουν μια τάση. Οι ερευνητές στις εργασίες τους καταπιάνονται με τα ακόλουθα ζητήματα:

- Ποιος είναι ο πιο κατάλληλος τρόπος διδασκαλίας του προγραμματισμού σε αρχαρίους, ο διαδικαστικός ή ο αντικειμενοστραφής;
- Πότε η μετάβαση από τον ένα τρόπο στο άλλο είναι πιο εύκολη; Από τον διαδικαστικό στον αντικειμενοστραφή ή αντίστροφα;

Πρόσφατες σχετικά έρευνες ανέδειξαν με λεπτομέρεια τα προβλήματα που αντιμετωπίζουν οι αρχάριοι χρήστες που διδάσκονται αντικειμενοστραφή προγραμματισμό. Ο Hadjerrouit (1999) αναφέρεται στον αντικειμενοστραφή προγραμματισμό και συμπεραίνει ότι η εκμάθησή του είναι δύσκολη για τους αρχάριους. Θεωρεί ότι η δυσκολία πηγάζει από το γεγονός ότι ο νέος αυτός τρόπος προγραμματισμού απαιτεί διαφορετικό τρόπο σκέψης και παρουσιάζει υψηλότερες απαιτήσεις στις διαδικασίες της ανάλυσης και της σχεδίασης του προγράμματος.

Οι Wiedenbeck et al (1999) μελέτησαν την κατανόηση διαδικαστικών και αντικειμενοστραφών προγραμμάτων σε φοιτητές Πανεπιστημίου, στο δεύτερο εξάμηνο των σπουδών τους. Οι φοιτητές διδάσκονταν είτε Pascal είτε C++ και αξιολογήθηκαν σε προγράμματα της αντίστοιχης γλώσσας. Τα προγράμματα σχεδιάστηκαν με τέτοιο τρόπο ώστε να είναι ισοδύναμα και για τις δύο γλώσσες. Σε μικρά σχετικά προγράμματα (που είχαν μία κλάση στη C++) δεν παρατηρήθηκε σημαντική διαφορά στη κατανόηση τους, αν και τα αντικειμενοστραφή τμήματα ήταν πιο κοντά στη κατανόηση της λειτουργίας του προγράμματος. Σε μεγαλύτερα προγράμματα (που περιλάμβαναν αρκετές κλάσεις στη C++) τα αποτελέσματα ήταν τελείως διαφορετικά. Οι διαδικαστικοί προγραμματιστές παρουσίασαν πολύ καλύτερα αποτελέσματα από τους αντικειμενοστραφείς προγραμματιστές σε όλες τις μετρήσεις. Οι ερευνητές κατέληξαν στο συμπέρασμα (Wiedenbeck et al, 1999:

"η κατανεμημένη φύση του ελέγχου ροής και λειτουργιών σε ένα αντικειμενοστραφές πρόγραμμα μπορεί να καταστήσει πιο δύσκολο για τους αρχάριους το σχηματισμό μιας νοητικής αναπαράστασης της λειτουργίας και του ελέγχου ροής ενός αντικειμενοστραφούς προγράμματος σε σχέση με το αντίστοιχο διαδικαστικό… Καταλήγουμε στο συμπέρασμα ότι οι δυσκολίες κατανόησης, που συνάντησαν οι αρχάριοι με τα μεγάλα αντικειμενοστραφή προγράμματα, αποδίδεται σε πρώτο βαθμό στο μεγαλύτερο χρόνο εκμάθησης του αντικειμενοστραφούς προγραμματισμού και σε δεύτερο βαθμό στη φύση των ίδιων των μεγάλων αντικειμενοστραφών προγραμμάτων."

Τα παραπάνω συμπεράσματα έρχονται σε αντίθεση με τη γενική αίσθηση ότι ο αντικειμενοστραφής προγραμματισμός είναι πιο κοντά στο φυσικό τρόπο αντίληψης και μοντελοποίησης πραγματικών καταστάσεων. Όπως επισημαίνουν και οι συγγραφείς (Wiedenbeck et al, 1999):

"Τα αποτελέσματα αυτά υποδηλώνουν ότι οι αρχάριοι αντικειμενοστραφείς προγραμματιστές επικεντρώνονται στο προγραμματιστικό μοντέλο, σε αντίθεση με την υπάρχουσα γενική αίσθηση ότι το αντικειμενοστραφές πρότυπο επικεντρώνει τον προγραμματιστή στο πεδίο του προβλήματος, μοντελοποιώντας το με ακρίβεια στον κώδικα του προγράμματος."

Σε παρόμοια συμπεράσματα κατέληξαν και οι Wiedenbeck & Ramalingam (1999) σε έρευνα για την κατανόηση μικρών προγραμμάτων σε C και C++. Πάλι και εδώ δεν παρουσιάστηκε καμία διαφορά στις μετρήσεις για την κατανόηση των προγραμμάτων. Σε ειδικές μετρήσεις όμως παρατηρήθηκε ότι οι φοιτητές ανέπτυξαν ισχυρές αναπαραστάσεις των μικρών αντικειμενοστραφών προγραμμάτων όσον αφορά τη λειτουργία του προγράμματος, ενώ ανέπτυξαν αδύναμες αναπαραστάσεις όσον αφορά τον έλεγχο ροής και άλλες διαδικασίες σχετικές με το πρόγραμμα. Αντίθετα στα μικρά διαδικαστικά προγράμματα οι φοιτητές ανέπτυξαν ισχυρές αναπαραστάσεις όσον αφορά τον έλεγχο ροής και άλλες διαδικασίες σχετικές με το πρόγραμμα.

Στο ερώτημα της μετάβασης από τον ένα τρόπο προγραμματισμού στον άλλο και την ευκολία ή δυσκολία που αυτό επιτυγχάνεται, οι απόψεις διίστανται. Διαφαίνεται πάντως μια τάση όλο και περισσότερο να επικρατήσει η άποψη ότι οι φοιτητές αντιμετωπίζουν πιο δύσκολα τη μετάβαση από τον διαδικαστικό προγραμματισμό στον αντικειμενοστραφή. Είναι γεγονός ότι ο αντικειμενοστραφής προγραμματισμός απαιτεί ένα διαφορετικό τρόπο προσέγγισης του προβλήματος, που δεν είναι τόσο άμεσα κατανοητός σε φοιτητές με προϋπάρχουσα εμπειρία στο διαδικαστικό προγραμματισμό (Hadjerrouit, 1999). Στην έρευνα που έκανε ο Hadjerrouit (1999) διαπίστωσε ότι οι φοιτητές με διαδικαστικό υπόβαθρο δυσκολεύτηκαν να κατανοήσουν αντικειμενοστραφείς έννοιες με αποτέλεσμα να συγχέουν τις μεθόδους με διαδικασίες και να μη μπορούν να αξιοποιήσουν πλήρως τις δυνατότητες του αντικειμενοστραφούς προγραμματισμού.

Οι Brilliant & Wiseman (1996) μελέτησαν τις απαντήσεις σε ερωτηματολόγια που έδωσαν διδάσκοντες σε τμήματα Επιστήμης Υπολογιστών και κατέληξαν σε διιστάμενα αποτελέσματα. Πιο συγκεκριμένα συμπεραίνουν ότι ούτε η επιλογή της γλώσσας ούτε η επιλογή του τρόπου – προτύπου προγραμματισμού, είτε διαδικαστικό είτε αντικειμενοστραφές είναι αυτό, έχει κάποια επίδραση στην επίδοση των μαθητών.

ΟΙ «ΕΜΠΟΡΙΚΕΣ» ΛΥΣΕΙΣ

Οι εμπορικές λύσεις, τα περιβάλλοντα προγραμματισμού και οι γλώσσες που διατίθενται εμπορικά, έχουν συγκεκριμένο στόχο, να ικανοποιήσουν, όσο το δυνατό περισσότερο, τον επαγγελματία προγραμματιστή. Τα εν λόγω εμπορικά περιβάλλοντα δεν μπορούν να ικανοποιήσουν το μαθητή, το φοιτητή, το νέο χρήστη επειδή απαιτούν ένα επίπεδο βασικών γνώσεων που ο αρχάριος δεν είναι σε θέση να κατέχει.

Από την εμπειρία μας τα προβλήματα που παρουσιάζονται στη διαδικασία εκμάθησης του προγραμματισμού, χρησιμοποιώντας τα διαθέσιμα εμπορικά περιβάλλοντα, είναι:

- Τα μηνύματα λάθους που παράγουν οι μεταγλωττιστές δεν είναι κατατοπιστικά. Τις περισσότερες φορές είναι δυσνόητα και παραπλανητικά για αρχάριους προγραμματιστές. Στο ίδιο συμπέρασμα καταλήγουν και ανάλογες έρευνες που έχουν γίνει από την επιστημονική κοινότητα (Schorsch, 1995). Ένα κλασσικό παράδειγμα λάθους που οδηγεί σε παραπλανητικό μήνυμα είναι η περίπτωση στην οποία ένας προγραμματιστής παραλείψει το ερωτηματικό «;» από το τέλος μιας εντολής σε πρόγραμμα Pascal ή C/C++. Ο αντίστοιχος μεταγλωττιστής αναγνωρίζει το συντακτικό λάθος αλλά συνήθως αναφέρεται σε αυτό με αριθμό γραμμής διαφορετικό (μεταγενέστερο) από εκείνον όπου υπάρχει το πρόβλημα, μια και ο συντακτικός αναλυτής (parser) αντιλαμβάνεται το λάθος στο σημείο αυτό. Αυτό το είδος του λάθους γίνεται εύκολα αντιληπτό από τους έμπειρους επαγγελματίες προγραμματιστές αλλά ταυτόχρονα παραπλανά τους νέους χρήστες.
- Δεν υποστηρίζεται καθόλου ή υποστηρίζεται σε μικρό βαθμό ο έλεγχος κατά τη διάρκεια εκτέλεσης ενός προγράμματος (runtime checking) που θα βοηθούσε τους αρχαρίους να

ανακαλύψουν λογικά, επί το πλείστον, λάθη, που θα ήταν δύσκολο να εντοπιστούν. Οι εμπορικοί μεταγλωττιστές συνήθως δεν παράγουν κώδικα για ανίχνευση λαθών κατά την εκτέλεση του προγράμματος, όπως για παράδειγμα η διαίρεση με το 0, η αναφορά σε στοιχείο πίνακα εκτός ορίων, η χρήση μη επιτρεπτών τύπων δεδομένων των τελεσταίων σε πράξεις όπως η ακέραια διαίρεση (div) ή το υπόλοιπο της διαίρεσης (mod), αφού κάτι τέτοιο δεν ενδιαφέρει τους έμπειρους προγραμματιστές (για λόγους ταχύτερης εκτέλεσης του κώδικα).

- Η πολυπλοκότητα των παρεχόμενων πληροφοριών οδηγεί σε σύγχυση τους νέους χρήστες. Το πρόβλημα είναι ότι πολλά περιβάλλοντα αναπτύσσονται για καθαρά επαγγελματίες χρήστες και παρουσιάζουν μια πολύ μεγάλη, σχεδόν πλήρη, γκάμα δυνατοτήτων, χαρακτηριστικών και λειτουργιών. Πολλά από τα βοηθητικά χαρακτηριστικά που προσφέρουν τα εμπορικά περιβάλλοντα ενώ είναι χρήσιμα για τους επαγγελματίες είναι τελείως περιττά για τους αρχάριους χρήστες. Οι μαθητές χάνονται σε αυτά τα περιβάλλοντα, και το αποτέλεσμα μπορεί να είναι το ίδιο με την περίπτωση της διδασκαλίας του προγραμματισμού χωρίς χρήση προγραμματιστικού περιβάλλοντος.
- Πολλά σύγχρονα γραφικά περιβάλλοντα προγραμματισμού επικεντρώνονται στην εύκολη δημιουργία ενός γραφικού ενδιαμέσου (GUI). Ως αποτέλεσμα ο χρήστης περισσότερο ασχολείται με τη δημιουργία ενός όμορφου και ελκυστικού περιβάλλοντος της εφαρμογής του παρά με την ανάλυση και τον προγραμματισμό της ίδιας της εφαρμογής. Επιπρόσθετα δημιουργείται σύγχυση στο τι πραγματικά είναι ένα ολοκληρωμένο προγραμματιστικό περιβάλλον. Αποτελεί ένα περιβάλλον που προσφέρει εργαλεία σχεδίασης του γραφικού ενδιαμέσου ή ένα περιβάλλον που γίνεται ο προγραμματισμός μιας εφαρμογής; Θεωρούμε ότι υπάρχουν πιο χρήσιμα εργαλεία για την εκμάθηση του προγραμματισμού από τα σύγχρονα εμπορικά γραφικά περιβάλλοντα.
- Η ταχύτητα εκτέλεσης του παραγόμενου κώδικα δεν είναι στην πρώτη γραμμή του ενδιαφέροντος για τους νέους χρήστες. Αντίθετα οι περισσότεροι από αυτούς θα ήταν διατεθειμένοι να θυσιάσουν την ταχύτητα εκτέλεσης προκειμένου να εξασφαλίσουν απλούστερο περιβάλλον με κατανοητά μηνύματα λάθους και ευκολότερο εντοπισμό των συντακτικών και λογικών λαθών.
- Επίσης δεν πρέπει να είναι αμελητέο το κόστος απόκτησης αδειών χρήσης των εμπορικών λύσεων που, ακόμη και με τις μειωμένες ειδικές εκπαιδευτικές τιμές, αποτελεί τροχοπέδη στην υιοθέτηση τους από την εκπαιδευτική κοινότητα.
- Το συνοδευτικό υποστηρικτικό υλικό των εμπορικών περιβαλλόντων (εγχειρίδια χρήσης, βιβλία, βοήθεια on-line) είναι προσαρμοσμένο σε επαγγελματίες προγραμματιστές. Οποιαδήποτε στιγμή ο αρχάριος αναζητήσει κάποια πληροφορία μέσα στο διαθέσιμο υλικό, ενδέχεται να οδηγηθεί σε πλήθος πληροφοριών που στο μεγαλύτερο βαθμό δεν του είναι απαραίτητα και μπορεί να τον μπερδέψουν περισσότερο.
- Ακόμα περισσότερο, ορισμένοι από τους αρχάριους Έλληνες προγραμματιστές, έχουν να αντιμετωπίσουν το πρόβλημα της (αγγλικής) γλώσσας στο περιβάλλον διεπαφής (user interface) που διαθέτουν τα εμπορικά περιβάλλοντα προγραμματισμού.

Πολλοί εκπαιδευτικοί δεν συνηθίζουν να χρησιμοποιούν ένα ολοκληρωμένο προγραμματιστικό περιβάλλον για τη διδασκαλία του προγραμματισμού, γιατί δεν θεωρούν κάποιο κατάλληλο και χρήσιμο για τους μαθητές τους. Προτιμούν να ωθήσουν τους μαθητές να εργαστούν σε περιβάλλοντα όπως το DOS ή το UNIX (εκτελώντας τους μεταγλωττιστές από τη γραμμή εντολής). Αυτό έχει ως αποτέλεσμα τη σπατάλη σημαντικού χρόνου στην εκμάθηση των

εντολών του λειτουργικού σε βάρος της εκμάθησης του προγραμματισμού. Το αποτέλεσμα είναι ότι, λόγω ανυπαρξίας κατάλληλων εργαλείων, έχουμε απώλεια χρόνου και πολύτιμων ευκαιριών για τη καλύτερη διδασκαλία και την εκμάθηση του προγραμματισμού.

ΕΠΙΛΟΓΟΣ

Όπως ήδη αναφέρθηκε έχουν γίνει σημαντικές προσπάθειες για την ανάπτυξη εκπαιδευτικών γλωσσών προκειμένου να υποστηριχθεί τόσο η διδασκαλία του προγραμματισμού αλλά και να βοηθηθούν οι αρχάριοι στην εκμάθηση του προγραμματισμού. Εναλλακτικές προτάσεις έχουν παρουσιαστεί από τους ερευνητές είτε με το διαδικαστικό πρότυπο είτε με το αντικειμενοστραφές. Επίσης πληθώρα εμπορικών λύσεων διατίθεται στην αγορά. Δεν υπάρχει όμως μέχρι σήμερα μια γλώσσα που να αποτελεί ένα απόλυτα παραδεκτό πρότυπο εισαγωγής στον προγραμματισμό για αρχάριους χρήστες.

ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Allen, R.K., Grant, Douglas D., & Smith, R. (1996). “Using Ada as the first Programming language: A Retrospective”. In Proceedings of Software Engineering: Education & Practice, (SE:E&P’96), IEEE Computer Society Press.
2. Brilliant, S. S. & Wiseman T. R. (1996). “The First Programming Paradigm and Language Dilemma”, Proceedings of the 27th SIGCSE Technical Symposium on Computer Science Education, pp. 338-342.
3. Brusilovsky, P., Calabrese, E., Hvorecky, E., Kouchnirenko, A., & Miller, P. (1997). “Mini-languages: A Way to Learn Programming Principles”. In Education and Information Technologies, 2(1): 65-83.
4. Conway D. (1993). “Criteria and Consideration in the Selection of a First Programming Language”, Technical Report 93/192, Department of Computer Science, Monash University, December 1993.
5. Deek, F., Kimmell, H., (1999). “Status of computer science education in secondary schools: one state's perspective”. Computer Science Education 9,2.
6. Duke, R.. Salzman, E., Burmeister, J., Poon, J., Murray,L. (2000). “Teaching Programming To Beginners – Choosing The Language Is Just The First Step”, Proceedings of the Australasian Conference on Computing Education, pp. 79-86, December 2000, Melbourne Australia.
7. Freund, S., and Roberts, E. (1996). “THETIS: An ANSI C Programming Environment for Introductory Use”, SIGCSE Bulletin, February, Vol. 28(1), 300-304.
8. Hadjerrouit, S. (1999). “A Constructivist Approach to Object-Oriented Design and Programming”. iTiCSE’99, 6/99 Cracow, Poland, ACM.
9. Holt, R.C., Hume, J.N.P. (1984). “Introduction to Computer Science using the Turing Programming Language”, Prentice-Hall.
10. Kölling, M. and Rosenberg, J. (1996). “Blue - A Language for Teaching Object-Oriented Programming”, SIGCSE Bulletin, 28, March, 190-194.
11. Levy, S. (1995). “Computer Language Usage in CS1: Survey Results”, SIGCSE Bulletin, 27(3), pp. 21-26.
12. Pane, J., Myers, B. (2000). “The Influence of the Psychology of Programming on a Language Design: Project Status Report”. 12th Annual Workshop of the Psychology of Programming Interest Group, PPIG 2000, Corigliano Calabro, Italy. 10-13 Apr.
13. Papert, S. (1980). “Mindstorms: Children, Computers and Powerful Ideas”. Brighton, Sussex: Harvester Press.
14. Pattis, R. E. (1981). “Karel - the robot, a gentle introduction to the art of programming”. Wiley, London.
15. Repenning, A. and Sumner, T. (1995). “Agentsheets: A Medium for Creating Domain-Oriented Visual Languages.” Computer 28: 17-25.

16. Schorsch, T. (1995). “CAP: An automated selfassessment tool to check Pascal programs for syntax, logic and style errors,” SIGCSE Bulletinj March.
17. Wiedenbeck, S., Ramalingam, V (1999) "Novice comprehension of small programs written in the procedural and object-oriented styles." Int. J. Hum.-Comput. Stud. 51(1): 71-87
18. Wiedenbeck, S., Ramalingam, V., Sarasamma, S., Corritore, C., (1999). “A comparison of the comprehension of object-oriented and procedural programs by novice programmers”. Interacting with Computers 11(3): 255-282
19. Wirth, N. & Jensen, K. (1975). “Pascal User Manual and Report”, Springer-Verlag.
20. Wirth, Niklaus (1988). “Programming in Modula-2”, 4th edition, Springer Verlag.