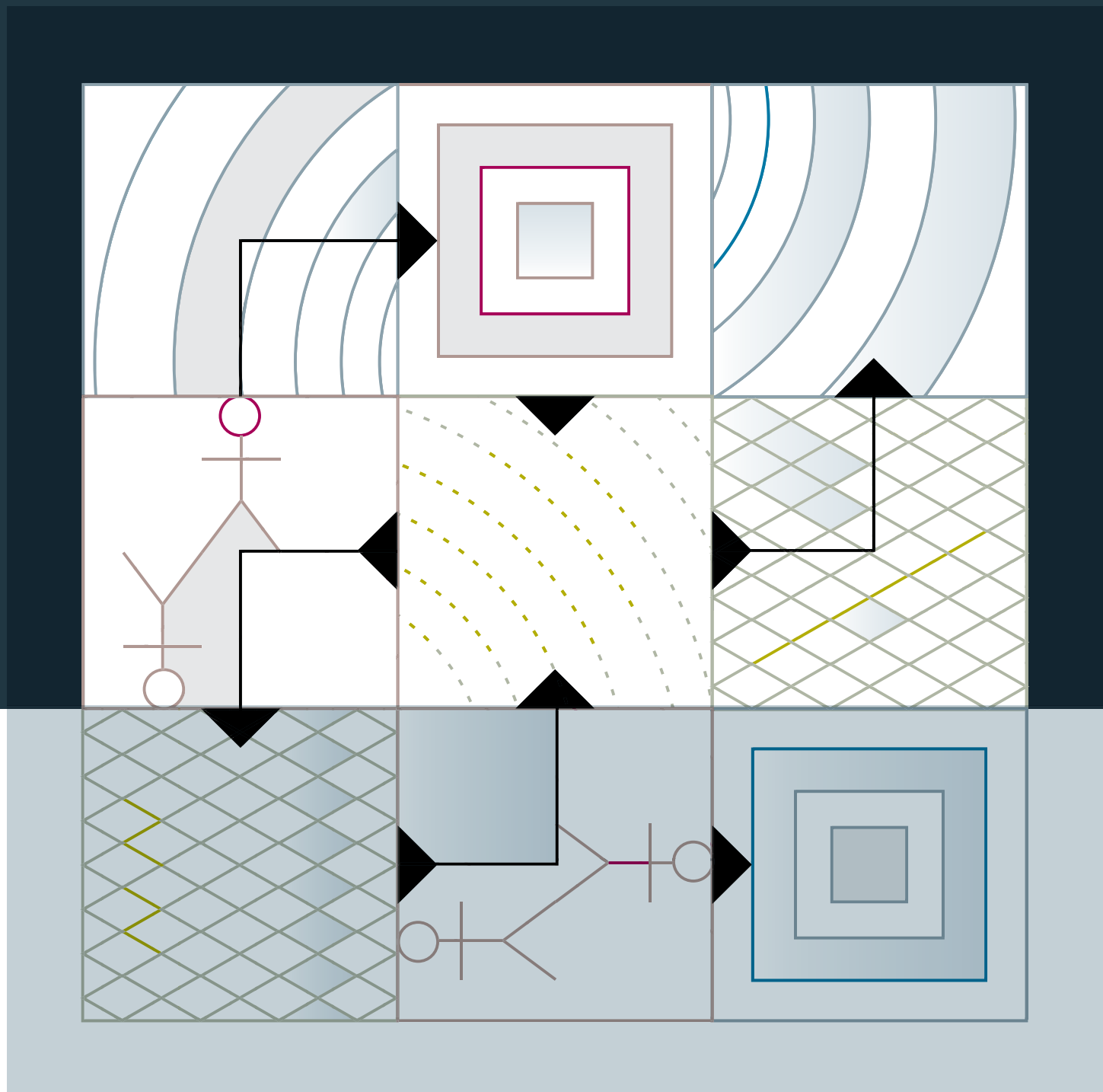


# ΣΤΟΙΧΕΙΑ ΤΕΧΝΟΛΟΓΙΑΣ ΛΟΓΙΣΜΙΚΟΥ

ΒΑΣΙΛΕΙΟΣ ΒΕΣΚΟΥΚΗΣ



Ελληνικά Ακαδημαϊκά Ηλεκτρονικά  
Συγγράμματα και Βοηθήματα  
[www.kallipos.gr](http://www.kallipos.gr)

**HEALLINK**  
Σύνδεσμος Ελληνικών Ακαδημαϊκών Βιβλιοθηκών



Ευρωπαϊκή Ένωση  
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ  
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ  
επένδυση στην κοινωνία της γνώσης  
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ  
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ  
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης



ΕΣΠΑ  
2007-2013  
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

ΤΙΤΛΟΣ:

Στοιχεία Τεχνολογίας Λογισμικού

ΣΥΓΓΡΑΦΕΑΣ:

Βασίλειος Βεσκούκης

ΚΡΙΤΙΚΟΣ ΑΝΑΓΝΩΣΤΗΣ:

Δημήτρης Καλλές

ΓΛΩΣΣΙΚΗ ΕΠΙΜΕΛΕΙΑ:

Ιωάννα Αγγελοπούλου

ΓΡΑΦΙΣΤΙΚΗ ΕΠΙΜΕΛΕΙΑ ΚΑΙ ΜΕΤΑΤΡΟΠΗ ΣΕ ΕΡΥΒ:

Κλειώ Καπαντώνη

Copyright © ΣΕΑΒ, 2015



Το παρόν έργο αδειοδοτείται υπό τους όρους της άδειας Creative Commons Αναφορά Δημιουργού - Μη Εμπορική Χρήση - Όχι Παράγωγα Έργα 3.0.

Για να δείτε ένα αντίγραφο της άδειας αυτής επισκεφτείτε τον ιστότοπο

<https://creativecommons.org/licenses/by-nc-nd/3.0/gr/>

ISBN: 978-960-603-060-4

Σύνδεσμος Ελληνικών Ακαδημαϊκών Βιβλιοθηκών

Εθνικό Μετσόβιο Πολυτεχνείο

Ηρώων Πολυτεχνείου 9, 15780 Ζωγράφου

Φίλε αναγνώστη,

Το βιβλίο που κρατάς αποτελεί εξέλιξη μιας συγγραφικής και εκπαιδευτικής δουλειάς που μετρά ήδη περίπου 15 έτη, στο θεματικό αντικείμενο της Τεχνολογίας Λογισμικού, όπως τελικά αποδόθηκε στην ελληνική ο όρος «Software Engineering». Ο ελληνικός όρος «τεχνολογία» δεν αποδίδει επαρκώς το «engineering», ωστόσο χρησιμοποιείται έστω και με κάποια απόκλιση, όπως περίπου συμβαίνει με τη χρήση του όρου «Χημική Μηχανική» η οποία αν και ακριβής, δεν είναι ιδιαίτερα διαδεδομένη απόδοση του «Chemical Engineering». Προγενέστερες μορφές τμημάτων του βιβλίου αυτού χρησιμοποιήθηκαν ως εκπαιδευτικά εγχειρίδια σε προπτυχιακά και μεταπτυχιακά προγράμματα σπουδών του Ελληνικού Ανοικτού Πανεπιστημίου, καθώς και σε μαθήματα στο Πανεπιστήμιο Πειραιά και αλλού. Κατά τη 15ετή πορεία από την αρχική συγγραφή, χιλιάδες σπουδαστές έκριναν με μεγάλη αυστηρότητα όλο το υλικό και πολλές διατυπώσεις ή προσεγγίσεις αποδείχθηκαν ατυχείς και χρειάστηκε να αλλάξουν. Επίσης, μεγάλο πλήθος ασκήσεων που εκπονήθηκαν από σπουδαστές με κύριο εγχειρίδιο το υλικό του βιβλίου, προσέφερε ανάδραση που στο μεγαλύτερο βαθμό ενσωματώθηκε στην παρούσα, ψηφιακή πλέον, έκδοση.

Το λογισμικό είναι εκείνο το συστατικό που, αν και το ίδιο δεν έχει χειροπιαστή υπόσταση, μπορεί να καταστήσει μια υπολογιστική και όχι μόνο μηχανή χρήσιμη στον άνθρωπο. Σήμερα ηλεκτρονικοί υπολογιστές βρίσκονται σε κάθε συσκευή που μπορούμε να διανοηθούμε – τα παραδείγματα είναι μάλλον περιττά. Όσο γενικεύεται αυτή η διείσδυση, τόσο περισσότερες γίνονται οι απαιτήσεις μας και σύνθετες οι εργασίες που τους αναθέτουμε. Η ικανοποίηση των απαιτήσεων αυτών γίνεται με τη βοήθεια του λογισμικού, η πολυπλοκότητα του οποίου - αναπόφευκτα - συνεχώς και αυξάνεται.

Αν συνδυάσει κανείς το γεγονός ότι το λογισμικό λειτουργεί σε υπολογιστικές μηχανές οι οποίες συνεχώς εξελίσσονται και ότι ικανοποιεί απαιτήσεις οι οποίες γίνονται ολοένα περισσότερες, πιο πολύπλοκες και μεταβάλλονται ταχύτατα με το χρόνο, με την μη χειροπιαστή φύση του λογισμικού, τότε μπορεί εύκολα να υποψιαστεί ότι η κατασκευή του είναι από μόνη της μια ιδιαίτερα δύσκολη υπόθεση. Πράγματι, από τα πρώτα χρόνια της διάδοσης των υπολογιστών, όχι ακόμα σε ευρεία κλίμακα, εκδηλώθηκαν σημαντικά

προβλήματα στην κατασκευή λογισμικού. Είναι χαρακτηριστικό ότι ο όρος «Τεχνολογία Λογισμικού» (Software Engineering) εισήχθη για πρώτη φορά μαζί με τον όρο «Κρίση Λογισμικού» (Software Crisis) το 1968.

Οι πρωτοπόροι στην έρευνα και στην πρακτική της ανάπτυξης λογισμικού δεν μπορούσαν να φανταστούν ούτε πόσο γρήγορα θα μεγάλωναν σε εύρος και βάθος οι απαιτήσεις των χρηστών των υπολογιστικών συστημάτων, ούτε ποια μορφή θα έχουν οι υπολογιστές στο μέλλον. Σήμερα η πραγματικότητα έχει ξεπεράσει κάθε επιστημονική φαντασία. Η αλληλεπίδραση των χρηστών με το λογισμικό δε γίνεται με λίγες και μεγάλες εφαρμογές, όπως στο παρελθόν, αλλά με πολλές και μικρές. Μάλιστα, οι εφαρμογές λογισμικού που χρησιμοποιούμε ολοένα και συχνότερα βρίσκονται διάσπαρτες σε κάποιο δίκτυο, που δεν το γνωρίζουμε και το αποκαλούμε «σύννεφο» (cloud). Στο σύννεφο πρόκειται να συνυπάρξουν ενοποιημένα στο κοντινό μέλλον όλα τα πληροφοριακά και επικοινωνιακά συστήματα.

Έτσι, οι κατασκευαστές λογισμικού και οι ερευνητές προσπαθούν να προτείνουν τρόπους ώστε να γίνεται σωστά και αποτελεσματικά η κατασκευή λογισμικού καλής ποιότητας. Πολλές «νέες τεχνολογίες» για το ίδιο το λογισμικό εισήχθησαν κατά καιρούς και εισάγονται συνεχώς. Συχνά, οι όροι της αγοράς και του ανταγωνισμού σκιάζουν τις καθαρά τεχνικές όψεις αυτών, αλλά κάτι τέτοιο δεν είναι πρωτόγνωρο. Σε κάθε εποχή, ο ενθουσιασμός και οι τυμπανοκρουσίες της έλευσης μιας νέας προσέγγισης, έδιναν αργά ή γρήγορα τη θέση τους στην προσγειωμένη πραγματικότητα. Τα προβλήματα στην ανάπτυξη του λογισμικού συμπεριφέρονταν λίγο ως πολύ ως «λερναία ύδρα», όπου στη θέση κάθε κεφαλιού που κόβονταν, φύτρωναν περισσότερα. Σήμερα, η αναζήτηση του «καλύτερου» τρόπου κατασκευής λογισμικού θεωρείται ιδεατή επιδίωξη. Έχει καταστεί σαφές ότι δεν υπάρχει καμία «χρυσή συνταγή» και ότι η ανάπτυξη του λογισμικού οφείλει να είναι μια ιδιαίτερα ευέλικτη διαδικασία, εύκολα προσαρμόσιμη στις εκάστοτε συνθήκες, αλλά και στη φύση του εκάστοτε προβλήματος στην επίλυση του οποίου χρησιμοποιείται λογισμικό.

Η κάποτε καινοτομική «Δομημένη Ανάλυση και Σχεδίαση» έδωσε τη θέση της στην «Αντικειμενοστρεφή Τεχνολογία» η οποία αντιμετωπίζει με εντελώς διαφορετικό τρόπο την προσέγγιση του ίδιου προβλήματος: ποια

δομικά συστατικά λογισμικού να κατασκευάσουμε για να ικανοποιήσουμε ένα γνωστό (;) σύνολο απαιτήσεων των χρηστών, με το μικρότερο κόστος, στον συντομότερο χρόνο και καλύτερα από τον ανταγωνισμό. Η αντικειμενοστρεφής τεχνολογία είναι υπερσύνολο της δομημένης προσέγγισης. Για το λόγο αυτό η δομημένη ανάλυση και σχεδίαση δεν αντιμετωπίζεται στο βιβλίο αυτό σαν παρωχημένη γνώση, αλλά χρησιμοποιείται και στην πράξη ως εργαλείο δόμησης των «ενδότερων» συστατικών στοιχείων του λογισμικού. Η φιλοδοξία και συνάμα η πρόκληση που έχει να αντιμετωπίσει η Τεχνολογία Λογισμικού είναι να περιγράψει διαδικασίες που να είναι τεκμηριωμένες, σαφείς, προσαρμόσιμες στις εκάστοτε συνθήκες, εύκολα εφαρμόσιμες και οι οποίες οδηγούν στην κατασκευή καλής ποιότητας λογισμικού, μέσα στο προκαθορισμένο χρονοδιάγραμμα και προϋπολογισμό – πράγματα προφανώς αντικρουόμενα. Είναι, επίσης, να βοηθήσει τον μηχανικό λογισμικού να δει τη μεγάλη εικόνα, το δάσος και όχι το δέντρο, ώστε να μπορέσει να αναπτύξει αυτά που έχουν νόημα, προσπερνώντας τις εφήμερες λεπτομέρειες των εναλλασσόμενων εικόνων της αγοράς.

Όπως και σε άλλα τεχνικά έργα, η ικανοποίηση όλων των απαιτήσεων δεν είναι εύκολη. Η επιδίωξη καλύτερης ποιότητας και πληρότητας σε ένα τεχνικό έργο (όπως για παράδειγμα ένας αυτοκινητόδρομος ή μία γέφυρα) είναι αναμενόμενο ότι και θα το καθυστερήσει και θα καταναλώσει μεγαλύτερο προϋπολογισμό. Σκεφτείτε πόσο μεγαλύτερο μπορεί να γίνει το πρόβλημα αυτό, όταν το τεχνικό έργο είναι μη χειροπιαστό και κατασκευάζεται με απαιτήσεις που μπορεί να μεταβάλλονται κατά τη διάρκεια της κατασκευής του.

Σε ένα άλλο, πιο σημαντικό μήκος κύματος, είναι σκόπιμο να τονίσουμε μια ουσιώδη διαφορά του τεχνικού έργου ανάπτυξης λογισμικού από την κατασκευή ενός οποιουδήποτε κλασσικού τεχνικού έργου: για το λογισμικό το μόνο απαιτούμενο κεφάλαιο είναι η ανθρώπινη διάνοια. Για να γίνει κανείς κατασκευαστής λογισμικού δεν απαιτούνται τα κεφάλαια που απαιτούνται για να γίνει κατασκευαστής δημόσιων τεχνικών έργων. Αυτό ισχύει και σε μακρο-οικονομικό επίπεδο: για να γίνει μια χώρα ισχυρή στη βιομηχανία λογισμικού δεν είναι απαραίτητο να κάνει άλλες επενδύσεις παρά μόνο εκείνες που σχετίζονται με την ανάπτυξη του ανθρώπινου δυναμικού. Για χώρες μικρές όπως η δική μας, αυτό δίνει στην Τεχνολογία Λογισμικού μια άλλη διάσταση



πρόκλησης: αν αναπτύξουμε μια ισχυρή βιομηχανία λογισμικού μπορούμε να διεκδικήσουμε μια καλύτερη θέση στην παγκοσμιοποιημένη οικονομία, την «έξοδο από την κρίση» και άλλα συναφή. Αυτά είναι δυστυχώς «ψιλά γράμματα» για ανθρώπους που αποκαλούν την παραβίαση των νόμων της φύσης «πολιτική απόφαση», είναι όμως σκόπιμο να τα υπενθυμίζει κανείς.

Ένα μικρό λιθαράκι στην ανάπτυξη του ανθρώπινου παράγοντα που θα μπορέσει να μας οδηγήσει σε μια τέτοια πορεία, φιλοδοξεί να βάλει η γνώση που είναι αποτυπωμένη στο ανά χείρας, ή κυριολεκτικά στο «εντός της συσκευής ανάγνωσης» που χρησιμοποιείς, φίλε αναγνώστη. Είναι προϊόν ακαδημαϊκής και επαγγελματικής εμπειρίας στο χώρο της ανάπτυξης λογισμικού και επιχειρείται να σου δοθεί με τρόπο σαφή και κατανοητό. Σε ορισμένες περιπτώσεις, στη βιβλιογραφία ίσως να συναντήσεις ελαφρώς διαφοροποιημένες προσεγγίσεις ή ακόμη και ορισμούς του ίδιου όρου. Αυτό είναι ενδεικτικό της κατάστασης σύγχυσης που επικράτησε για πολλά χρόνια στην κοινότητα των κατασκευαστών λογισμικού. Σύντομα θα μπορείς να διακρίνεις το ουσιώδες από το δευτερεύον, σύντομα θα αντιλαμβάνεσαι την ουσία και όχι τον τύπο των ορισμών.

Το ζητούμενο, ούτως ή άλλως, δεν είναι να απομνημονεύσεις ούτε να αποδεχτείς γνώση «άνευ όρων». Είναι να μάθεις πώς να μαθαίνεις, να κρίνεις και να αξιολογείς. Οι γνώσεις που περιέχουν τα βιβλία πληροφορικής σύντομα καθίστανται ανεπίκαιρες. Αυτό που ίσως μένει, είναι η συμβολή τους στη διαμόρφωση ενός τρόπου σκέψης που σε βοηθάει να βλέπεις το δάσος μέσα στο οικοσύστημά του και όχι κάθε δέντρο ξεχωριστά. Αυτή είναι και η δική μας επιδίωξη.

Επιθυμώ να αφιερώσω το βιβλίο στους μαχόμενους πληροφορικάριους, σε όλους εκείνους που δημιουργούν και δεν διαχειρίζονται. Λυπούμαι για λογαριασμό εκείνων που κάποτε πέταξαν το λογισμικό που είχαν δημιουργήσει και το αντικατέστησαν με «εισαγόμενο» με σκοπό να γίνουν «πιο ελκυστικοί» στο χρηματιστήριο. Είδαμε πού οδήγησαν τέτοιες επιλογές την κατά τα άλλα «περήφανη για τα μυαλά της» χώρα μας.

*Βασίλειος Βεσκούκης*

*Αθήνα, 2015*

# ΠΕΡΙΕΧΟΜΕΝΑ

## Αποδόσεις όρων / συντομεύσεις

---

## ΚΕΦΑΛΑΙΟ 1

---

ΓΝΩΡΙΜΙΑ ΜΕ ΤΗΝ ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ	8
ΕΝΟΤΗΤΑ 1.1.      ΥΠΟΛΟΓΙΣΤΕΣ ΚΑΙ ΛΟΓΙΣΜΙΚΟ	9
ΕΝΟΤΗΤΑ 1.2.      ΤΕΧΝΙΚΕΣ ΚΑΤΑΣΚΕΥΕΣ ΚΑΙ ΛΟΓΙΣΜΙΚΟ	11
ΕΝΟΤΗΤΑ 1.3.      ΚΡΙΣΗ ΛΟΓΙΣΜΙΚΟΥ	13
ΕΝΟΤΗΤΑ 1.4.      ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ	15
ΕΝΟΤΗΤΑ 1.5.      ΤΟ ΛΟΓΙΣΜΙΚΟ ΩΣ ΜΕΡΟΣ ΣΥΣΤΗΜΑΤΩΝ	17
ΕΝΟΤΗΤΑ 1.6.      ΤΟ ΛΟΓΙΣΜΙΚΟ ΩΣ ΠΡΟΪΟΝ	21
ΕΝΟΤΗΤΑ 1.7.      ΣΥΣΤΑΤΙΚΑ ΣΤΟΙΧΕΙΑ ΛΟΓΙΣΜΙΚΟΥ	24
ΕΝΟΤΗΤΑ 1.8.	27
ΕΝΟΤΗΤΑ 1.9.      ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ	28
ΒΙΒΛΙΟΓΡΑΦΙΑ	33

ΚΕΦΑΛΑΙΟ 2

ΜΟΝΤΕΛΑ ΚΥΚΛΟΥ ΖΩΗΣ ΛΟΓΙΣΜΙΚΟΥ		34
ΕΝΟΤΗΤΑ 2.1.	Η ΕΝΝΟΙΑ ΤΟΥ ΜΟΝΤΕΛΟΥ ΚΥΚΛΟΥ ΖΩΗΣ	37
ΕΝΟΤΗΤΑ 2.2.	ΤΟ ΜΟΝΤΕΛΟ ΤΟΥ ΚΑΤΑΡΡΑΚΤΗ	46
ΕΝΟΤΗΤΑ 2.3.	ΤΟ ΜΟΝΤΕΛΟ ΠΡΩΤΟΤΥΠΟΠΟΙΗΣΗΣ	49
ΕΝΟΤΗΤΑ 2.4.	ΤΟ ΜΟΝΤΕΛΟ ΛΕΙΤΟΥΡΓΙΚΗΣ ΕΠΑΥΞΗΣΗΣ	52
ΕΝΟΤΗΤΑ 2.5.	ΤΟ ΣΠΕΙΡΟΕΙΔΕΣ ΜΟΝΤΕΛΟ	55
ΕΝΟΤΗΤΑ 2.6.	ΤΟ ΜΟΝΤΕΛΟ ΤΟΥ ΠΙΔΑΚΑ	58
ΕΝΟΤΗΤΑ 2.7.	ΣΥΓΧΡΟΝΑ ΜΟΝΤΕΛΑ ΚΥΚΛΟΥ ΖΩΗΣ ΛΟΓΙΣΜΙΚΟΥ	61
ΕΝΟΤΗΤΑ 2.8.	ΣΥΓΧΡΟΝΑ ΜΟΝΤΕΛΑ ΚΥΚΛΟΥ ΖΩΗΣ ΛΟΓΙΣΜΙΚΟΥ	64
ΕΝΟΤΗΤΑ 2.9.	ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ	69
ΒΙΒΛΙΟΓΡΑΦΙΑ		72



# ΚΕΦΑΛΑΙΟ 3

ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ ΔΙΑΤΑΞΕΙΣ ΛΟΓΙΣΜΙΚΟΥ	73
<hr/>	
ΕΝΟΤΗΤΑ 3.1.     Η ΕΝΝΟΙΑ ΤΗΣ ΔΙΑΤΑΞΗΣ ΛΟΓΙΣΜΙΚΟΥ	75
ΕΝΟΤΗΤΑ 3.2.     ΤΥΠΙΚΕΣ ΔΙΑΤΑΞΕΙΣ ΛΟΓΙΣΜΙΚΟΥ	78
3.2.1.   Η μονολιθική διάταξη	
3.2.2.   Η διάταξη πελάτη-εξυπηρετητή	
3.2.3.   Η τριμερής διάταξη	
3.2.4.   Η πολυμερής διάταξη	
ΒΙΒΛΙΟΓΡΑΦΙΑ	88

## ΚΕΦΑΛΑΙΟ 4

### ΠΡΟΔΙΑΓΡΑΦΗ ΑΠΑΙΤΗΣΕΩΝ ΑΠΟ ΤΟ ΛΟΓΙΣΜΙΚΟ 89

---

#### ΕΝΟΤΗΤΑ 4.1. Η ΕΝΝΟΙΑ ΤΗΣ ΑΠΑΙΤΗΣΗΣ ΑΠΟ ΤΟ ΛΟΓΙΣΜΙΚΟ 91

4.1.1. Απαιτήσεις από το σύστημα

4.1.2. Τι είναι «απαίτηση από το λογισμικό»;

4.1.3. Πώς ταξινομούνται οι απαιτήσεις από το λογισμικό;

#### ΕΝΟΤΗΤΑ 4.2. ΜΗΧΑΝΙΚΗ ΑΠΑΙΤΗΣΕΩΝ 106

4.2.1. Εισαγωγή

4.2.2. Βήματα στον προσδιορισμό απαιτήσεων

#### ΕΝΟΤΗΤΑ 4.3. ΑΝΑΛΥΣΗ ΚΑΙ ΠΡΟΔΙΑΓΡΑΦΗ ΑΠΑΙΤΗΣΕΩΝ 111

4.3.1. Ανάλυση απαιτήσεων

4.3.2. Προδιαγραφή απαιτήσεων

#### ΕΝΟΤΗΤΑ 4.4. ΚΑΤΑΓΡΑΦΗ ΤΩΝ ΑΠΑΙΤΗΣΕΩΝ ΑΠΟ ΤΟ ΛΟΓΙΣΜΙΚΟ 122

4.4.1. Εισαγωγή

4.4.2. Διαγράμματα ροής δεδομένων

4.4.3. Διαγράμματα οντοτήτων – συσχετίσεων

4.4.4. Διαγράμματα μετάβασης καταστάσεων

4.4.5. Το λεξικό δεδομένων

#### ΕΝΟΤΗΤΑ 4.5. ΠΡΟΒΛΗΜΑΤΑ ΣΤΟΝ ΠΡΟΣΔΙΟΡΙΣΜΟ ΑΠΑΙΤΗΣΕΩΝ 163

4.5.1. Προβλήματα επικοινωνίας

4.5.2. Προβλήματα προτύπων

4.5.3. Προβλήματα γλώσσας

4.5.4. Προβλήματα οικονομικά

<b>ΕΝΟΤΗΤΑ 4.6.</b>	<b>ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ ΑΥΤΟΑΞΙΟΛΟΓΗΣΗΣ</b>	<b>169</b>
<b>4.6.1.</b>	<b>Δραστηριότητες</b>	
<b>4.6.2.</b>	<b>Ασκήσεις αυτοαξιολόγησης</b>	
<b>ΒΙΒΛΙΟΓΡΑΦΙΑ</b>		<b>183</b>

# ΚΕΦΑΛΑΙΟ 5

## ΔΟΜΗΜΕΝΗ ΣΧΕΔΙΑΣΗ 184

---

ΕΝΟΤΗΤΑ 5.1. ΣΚΟΠΟΣ ΤΗΣ ΣΧΕΔΙΑΣΗΣ 186

ΕΝΟΤΗΤΑ 5.3. ΤΕΧΝΟΤΡΟΠΙΕΣ ΣΧΕΔΙΑΣΗΣ 189

5.2.1. Δομημένη σχεδίαση

5.2.2. Αντικειμενοστρεφής σχεδίαση

ΕΝΟΤΗΤΑ 5.2. ΑΝΤΙΚΕΙΜΕΝΟ ΚΑΙ ΑΠΟΤΕΛΕΣΜΑΤΑ ΤΗΣ ΣΧΕΔΙΑΣΗΣ 192

5.3.1. Εισαγωγή

5.3.2. Αρχιτεκτονική σχεδίαση.

5.3.3. Σχεδίαση διεπαφών

5.3.4. Λεπτομερής σχεδίαση μονάδων

5.3.5. Σχεδίαση δεδομένων

5.3.6. Το έγγραφο περιγραφής του σχεδίου του λογισμικού

ΕΝΟΤΗΤΑ 5.4. ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΣΧΕΔΙΑΣΗ 206

5.4.1. Ορισμοί

5.4.2. Βήματα κατασκευής διαγραμμάτων δομής

ΕΝΟΤΗΤΑ 5.5. ΛΕΠΤΟΜΕΡΗΣ ΣΧΕΔΙΑΣΗ ΜΟΝΑΔΩΝ 239

ΕΝΟΤΗΤΑ 5.6. ΣΧΕΔΙΑΣΗ ΔΕΔΟΜΕΝΩΝ 247

ΕΝΟΤΗΤΑ 5.7. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ  
ΑΥΤΟΑΞΙΟΛΟΓΗΣΗΣ 249

5.7.1. Δραστηριότητες

5.7.2. Ασκήσεις αυτοαξιολόγησης

ΒΙΒΛΙΟΓΡΑΦΙΑ 266

# ΚΕΦΑΛΑΙΟ 6

## ΠΑΡΑΓΩΓΗ ΠΗΓΑΙΟΥ ΚΩΔΙΚΑ 267

---

### ΕΝΟΤΗΤΑ 6.1. ΑΠΟ ΤΗ ΣΧΕΔΙΑΣΗ ΣΤΗΝ ΚΩΔΙΚΟΠΟΙΗΣΗ 269

6.1.1. Λογισμικό χωρίς σφάλματα

6.1.2. Εργαλεία κωδικοποίησης

### ΕΝΟΤΗΤΑ 6.2. ΕΠΙΘΥΜΗΤΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΟΥ ΠΗΓΑΙΟΥ ΚΩΔΙΚΑ 274

6.2.1. Επάρκεια

6.2.2. Επίδοση

6.2.3. Αναγνωσιμότητα

6.2.4. Τεκμηρίωση

6.2.5. Μεταφερσιμότητα

6.2.6. Δυνατότητα επαναχρησιμοποίησης

### ΕΝΟΤΗΤΑ 6.3. ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ 278

6.3.1. Εισαγωγή

6.3.2. Δομημένος προγραμματισμός

6.3.3. Χαρακτηριστικά σύγχρονων γλωσσών προγραμματισμού

### ΕΝΟΤΗΤΑ 6.4. ΤΕΧΝΙΚΕΣ ΣΥΓΓΡΑΦΗΣ ΠΗΓΑΙΟΥ ΚΩΔΙΚΑ 288

6.4.1. Τεχνικές αποφυγής σφαλμάτων

6.4.2. Ανοχή σε σφάλματα

6.4.3. Εντοπισμός και διόρθωση σφαλμάτων

### ΕΝΟΤΗΤΑ 6.5. ΕΠΑΝΑΧΡΗΣΙΜΟΠΟΙΗΣΗ ΜΟΝΑΔΩΝ ΠΡΟΓΡΑΜΜΑΤΟΣ 294

### ΕΝΟΤΗΤΑ 6.6. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ 295

6.6.1. Δραστηριότητες

## **ΚΕΦΑΛΑΙΟ 7**

### **ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗΣ ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ 300**

#### **ΕΝΟΤΗΤΑ 7.1. Η ΠΟΡΕΙΑ ΤΗΣ ΔΟΜΗΜΕΝΗΣ ΑΝΑΛΥΣΗΣ ΚΑΙ ΣΧΕΔΙΑΣΗΣ 303**

#### **ΕΝΟΤΗΤΑ 7.2. ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΟΥ ΣΥΓΧΡΟΝΟΥ ΛΟΓΙΣΜΙΚΟΥ 307**

#### **ΕΝΟΤΗΤΑ 7.3. ΕΞΕΛΙΞΕΙΣ ΚΑΙ ΤΑΣΕΙΣ ΣΤΗΝ ΑΝΑΠΤΥΞΗ ΤΟΥ ΛΟΓΙΣΜΙΚΟΥ 311**

#### **ΕΝΟΤΗΤΑ 7.4. ΑΔΥΝΑΜΙΕΣ ΤΗΣ ΔΟΜΗΜΕΝΗΣ ΑΝΑΛΥΣΗΣ ΚΑΙ ΣΧΕΔΙΑΣΗΣ 314**

#### **ΕΝΟΤΗΤΑ 7.5. ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ ΤΗΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΟΥΣ ΤΕΧΝΟΛΟΓΙΑΣ 317**

##### **7.5.1. Ορισμοί**

##### **7.5.2. Σχέσεις μεταξύ κλάσεων**

#### **ΕΝΟΤΗΤΑ 7.6. ΕΝΑΣ ΑΛΛΟΣ ΤΡΟΠΟΣ ΠΑΡΑΣΤΑΣΗΣ ΤΟΥ ΚΟΣΜΟΥ 338**

#### **ΕΝΟΤΗΤΑ 7.7. ΣΥΜΒΟΛΙΣΜΟΙ ΚΑΙ ΠΡΟΤΥΠΑ 344**

#### **ΕΝΟΤΗΤΑ 7.8. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ 349**

**ΒΙΒΛΙΟΓΡΑΦΙΑ**



# ΚΕΦΑΛΑΙΟ 8

ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗ ΑΝΑΛΥΣΗ	360
ΕΝΟΤΗΤΑ 8.1. ΕΝΑ ΓΕΝΙΚΟ ΠΛΑΙΣΙΟ ΓΙΑ ΤΗΝ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗ ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ	364
ΕΝΟΤΗΤΑ 8.2. Η ΕΝΝΟΙΑ ΤΗΣ ΠΕΡΙΠΤΩΣΗΣ ΧΡΗΣΗΣ	371
8.2.1. Εισαγωγή	
8.2.2. Τι είναι «περίπτωση χρήσης»;	
8.2.3. Πώς προδιαγράφεται μια περίπτωση χρήσης;	
8.2.4. Ένα σημείο αναφοράς	
ΕΝΟΤΗΤΑ 8.3. ΠΡΟΣΔΙΟΡΙΣΜΟΣ ΤΩΝ ΛΕΙΤΟΥΡΓΙΚΩΝ ΑΠΑΙΤΗΣΕΩΝ ΩΣ ΠΕΡΙΠΤΩΣΕΩΝ ΧΡΗΣΗΣ	393
ΕΝΟΤΗΤΑ 8.4. ΑΠΟ ΤΙΣ ΠΕΡΙΠΤΩΣΕΙΣ ΧΡΗΣΗΣ ΣΤΟ ΜΟΝΤΕΛΟ ΑΝΑΛΥΣΗΣ	407
8.4.1. Το μοντέλο ανάλυσης	
8.4.2. Κλάσεις στο μοντέλο ανάλυσης	
8.4.3. Πακέτα ανάλυσης	
ΕΝΟΤΗΤΑ 8.5. ΒΗΜΑΤΑ ΣΤΗΝ ΑΝΑΛΥΣΗ	415
8.5.1. Αρχιτεκτονική ανάλυση	
8.5.2. Ανάλυση περιπτώσεων χρήσης	
8.5.3. Ανάλυση κλάσεων	
8.5.4. Ανάλυση πακέτων	
ΕΝΟΤΗΤΑ 8.6. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ	443
ΒΙΒΛΙΟΓΡΑΦΙΑ	463

## ΚΕΦΑΛΑΙΟ 9

### ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗ ΣΧΕΔΙΑΣΗ 464

---

#### ΕΝΟΤΗΤΑ 9.1. Η ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗΣ ΣΧΕΔΙΑΣΗ ΣΤΟ ΜΟΝΤΕΛΟ ΚΥΚΛΟΥ ΖΩΗΣ 467

#### ΕΝΟΤΗΤΑ 9.2. ΤΟ ΜΟΝΤΕΛΟ ΣΧΕΔΙΑΣΗΣ 470

9.2.1. Οι περιπτώσεις χρήσης στο μοντέλο σχεδίασης

9.2.2. Πώς σχεδιάζεται η περίπτωση χρήσης

9.2.3. Διεπαφές στο μοντέλο σχεδίασης

9.2.4. Κλάσεις στο μοντέλο σχεδίασης

9.2.5. Αρχιτεκτονική όψη και υποσυστήματα

9.2.6. Μοντέλο διάταξης (Deployment)

#### ΕΝΟΤΗΤΑ 9.3. ΒΗΜΑΤΑ ΣΤΗ ΣΧΕΔΙΑΣΗ 492

9.3.1. Σχεδίαση περιπτώσεων χρήσης

9.3.2. Αρχιτεκτονική σχεδίαση και υποσυστήματα

9.3.3. Δημιουργία μοντέλου διάταξης

#### ΕΝΟΤΗΤΑ 9.4. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ 510

#### ΒΙΒΛΙΟΓΡΑΦΙΑ 534

# ΚΕΦΑΛΑΙΟ 10

ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ	535
ΕΝΟΤΗΤΑ 10.1. ΤΟ ΜΟΝΤΕΛΟ ΥΛΟΠΟΙΗΣΗΣ ΣΤΗΝ ΕΝΟΠΟΙΗΜΕΝΗ ΠΡΟΣΕΓΓΙΣΗ ΑΝΑΠΤΥΞΗΣ ΛΟΓΙΣΜΙΚΟΥ	537
10.1.1 Συστατικό λογισμικού (Component) στο μοντέλο υλοποίησης	
10.1.3 Υποσύστημα στο μοντέλο υλοποίησης	
10.1.4 Διεπαφές στο μοντέλο υλοποίησης	
ΕΝΟΤΗΤΑ 10.2. ΕΡΓΑΣΙΕΣ ΠΟΥ ΕΚΤΕΛΟΥΝΤΑΙ ΣΤΗΝ ΥΛΟΠΟΙΗΣΗ	545
ΕΝΟΤΗΤΑ 10.3. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ	551
ΒΙΒΛΙΟΓΡΑΦΙΑ	554

# ΚΕΦΑΛΑΙΟ 11

ΕΛΕΓΧΟΣ ΚΑΙ ΔΙΟΡΘΩΣΗ ΣΦΑΛΜΑΤΩΝ	556
ΕΝΟΤΗΤΑ 11.1. ΕΛΕΓΧΟΣ ΛΟΓΙΣΜΙΚΟΥ	558
ΕΝΟΤΗΤΑ 11.2. ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΤΟΥ ΕΛΕΓΧΟΥ	561
ΕΝΟΤΗΤΑ 11.3. ΤΕΧΝΙΚΕΣ ΕΛΕΓΧΟΥ	563
11.3.1. Στρατηγική του μαύρου κουτιού.	
11.3.2. Στρατηγική του γυάλινου κουτιού	

<b>ΕΝΟΤΗΤΑ 11.4. ΕΚΤΕΛΕΣΗ ΕΛΕΓΧΟΥ</b>	<b>582</b>
II.4.1. Έλεγχος μονάδας.	
II.4.2. Έλεγχος συνένωσης.	
II.4.3. Έλεγχος συστήματος.	
II.4.4. Έλεγχος αποδοχής.	
<b>ΕΝΟΤΗΤΑ 11.5. Ο ΕΛΕΓΧΟΣ ΣΤΗΝ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗ ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ</b>	<b>595</b>
II.5.1. Αστοχία (Defect)	
II.5.2. Περίπτωση ελέγχου (Test Case)	
II.5.3. Πλάνο ελέγχου (Test Plan)	
II.5.4. Αξιολόγηση ελέγχου (Test Evaluation)	
<b>ΕΝΟΤΗΤΑ 11.6. ΒΗΜΑΤΑ ΕΛΕΓΧΟΥ ΣΤΗΝ ΕΝΟΠΟΙΗΜΕΝΗ ΠΡΟΣΕΓΓΙΣΗ</b>	<b>607</b>
II.6.1. Δημιουργία πλάνου ελέγχου	
II.6.2. Σχεδίαση ελέγχου	
II.6.3. Υλοποίηση ελέγχου	
II.6.4. Εκτέλεση ελέγχου ολοκλήρωσης	
II.6.5. Εκτέλεση ελέγχου συστήματος	
<b>ΕΝΟΤΗΤΑ 11.7. ΑΝΑΦΟΡΕΣ ΕΛΕΓΧΟΥ</b>	<b>618</b>
<b>ΕΝΟΤΗΤΑ 11.8. ΔΙΟΡΘΩΣΗ ΣΦΑΛΜΑΤΩΝ</b>	<b>620</b>
<b>ΕΝΟΤΗΤΑ 11.9. ΕΡΓΑΛΕΙΑ ΕΛΕΓΧΟΥ</b>	<b>621</b>
<b>ΕΝΟΤΗΤΑ 11.10. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ ΑΥΤΟΑΞΙΟΛΟΓΗΣΗΣ</b>	<b>623</b>
<b>ΒΙΒΛΙΟΓΡΑΦΙΑ</b>	<b>635</b>

# ΚΕΦΑΛΑΙΟ 12

ΔΙΟΙΚΗΣΗ ΣΧΗΜΑΤΙΣΜΩΝ ΛΟΓΙΣΜΙΚΟΥ	636
ΕΝΟΤΗΤΑ 12.1. ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ	639
I2.1.1. Η έννοια του σχηματισμού λογισμικού	
I2.1.2. Η έννοια της βασικής γραμμής	
ΕΝΟΤΗΤΑ 12.2. ΔΙΟΙΚΗΣΗ ΣΧΗΜΑΤΙΣΜΩΝ ΛΟΓΙΣΜΙΚΟΥ	646
ΕΝΟΤΗΤΑ 12.3. ΕΡΓΑΣΙΕΣ ΔΙΟΙΚΗΣΗΣ ΣΧΗΜΑΤΙΣΜΩΝ ΛΟΓΙΣΜΙΚΟΥ	650
I2.3.1. Καθορισμός σχηματισμών	
I2.3.2. Έλεγχος μεταβολών σχηματισμών	
I2.3.3. Έλεγχος ποιότητας σχηματισμών	
I2.3.4. Έκθεση κατάστασης σχηματισμών	
ΕΝΟΤΗΤΑ 12.4. ΕΡΓΑΛΕΙΑ ΔΙΟΙΚΗΣΗΣ ΣΧΗΜΑΤΙΣΜΩΝ ΛΟΓΙΣΜΙΚΟΥ	665
ΕΝΟΤΗΤΑ 12.5. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΑΥΤΟΑΞΙΟΛΟΓΗΣΗΣ	668
ΒΙΒΛΙΟΓΡΑΦΙΑ	672

# ΚΕΦΑΛΑΙΟ 13

ΔΙΑΣΦΑΛΙΣΗ ΠΟΙΟΤΗΤΑΣ ΛΟΓΙΣΜΙΚΟΥ	673
ΕΝΟΤΗΤΑ 13.1. ΠΟΙΟΤΗΤΑ ΛΟΓΙΣΜΙΚΟΥ	676
13.1.1. Η έννοια του σχηματισμού λογισμικού	
ΕΝΟΤΗΤΑ 13.2. ΕΠΙΘΕΩΡΗΣΕΙΣ ΠΟΙΟΤΗΤΑΣ ΛΟΓΙΣΜΙΚΟΥ	683
ΕΝΟΤΗΤΑ 13.3. ΠΡΟΤΥΠΑ ΛΟΓΙΣΜΙΚΟΥ	689
13.3.1. Πρότυπα και διασφάλιση ποιότητας λογισμικού	
13.3.2. Πρότυπα τεκμηρίωσης λογισμικού	
ΒΙΒΛΙΟΓΡΑΦΙΑ	698



# Αποδόσεις όρων / συντομεύσεις

---

Αμυντικός προγραμματισμός	Defensive programming
Αναγνωσιμότητα	Readability
Ανάλυση	Analysis
Ανατροφοδότηση	Feedback
Ανοχή σε σφάλματα	Fault tolerance
Αντίγραφο ασφαλείας	Backup
Αντικείμενο	Object
Αντικειμενοστρεφής	Object-oriented
Από επάνω προς τα κάτω	Top-down
Αποθήκες πληροφοριών λογισμικού	Software repositories
Αποθήκη λογισμικού	Software repository
Απόκριση	Response
Απόκρυψη πληροφοριών	Information hiding
Αποφυγή σφάλματος	Fault avoidance
Άποψη, όψη, εμφάνιση	View
Αφαίρεση ως προς τα δεδομένα	Data abstraction
Αφαίρεση ως προς τους μετασχηματισμούς	Functional abstraction
Αφαιρετική	Abstract
Βασική γραμμή	Baseline
Βασισμένη-στο-web	Web-based
Γεγονός	Event

Γενίκευση	Generalization
Γεννήτορας προγραμμάτων	Program generator
Γλώσσα σχεδίασης/περιγραφής προγράμματος language	Program description language
Διάγραμμα	Diagram
Διάγραμμα	Diagram
Διάγραμμα ακολουθίας ή αλληλουχίας ή σειράς	Sequence diagram
Διάγραμμα διάταξης	Deployment diagram
Διάγραμμα δραστηριότητας	Activity diagram
Διάγραμμα επικοινωνίας	Communication diagram
Διάγραμμα συνεργασίας	Collaboration diagram
Διαδικασία	Procedure
Διαδικασία ανάπτυξης λογισμικού	Software process
Διαδικασία επίλυσης προβλημάτων	Problem solving process
Διάταξη λογισμικού	Software deployment
Διάταξη πελάτη-εξυπηρετητή	Client-server
Διαχείριση δεδομένων	Data management
Διεπαφή	Interface
Διερμηνέας	Interpreter
Δομημένη ανάλυση	Structured analysis
Δομημένος προγραμματισμός	Structured programming
Δοσοληψία	Transaction
Εγγραφή	Record
Έκδοση	Version

Εκδοχή	Instance
Εκδοχή (λογισμικού)	Release
Εκλέπτυνση σε διαδοχικά βήματα	Stepwise refinement
Ελαφρύς πελάτης	Thin client
Έλεγχος	Testing
Εντοπιστής σφαλμάτων	Debugger
Εξυπηρετητής δεδομένων	Database server
Εξυπηρετητής εφαρμογών	Application server
Εξυπηρετητής web	Web server
Επαλήθευση	Verification
Επαναχρησιμοποίηση	Reusability
Επάρκεια	Efficiency
Επίδοση	Performance
Επιθεώρηση	Review
Επικοινωνία με το χρήστη	User interface
Επικύρωση	Validation
Επιχειρησιακή λογική	Business logic
Εργαλεία ανάπτυξης λογισμικού	Software development /
CASE tools	
Ηλεκτρονικός υπολογιστής	H/Y
Θεματικό πεδίο εφαρμογής	Application domain
Ιδίωμα	Attribute
Καταγραφή, παρακολούθηση, εποπτεία	Tracking
Καταρράκτης	Waterfall

Κελυφοποίηση	Encapsulation
Κελυφοποίηση	Encapsulation
Κέντρο δοσοληψιών	Transaction centre
Κληρονομικότητα	Inheritance
Κλήση (συνάρτησης)	Call
Κοινωνία της πληροφορίας	Information society
Κρίση λογισμικού	Software crisis
Κύκλος ζωής λογισμικού	Software life cycle
Λειτουργικό σύστημα	Operating system
Λεξικό δεδομένων	Data dictionary
Λογικός προγραμματισμός	Logic programming
Μεθοδολογία ανάπτυξης λογισμικού	Software development methodology
Μεταφερισιμότητα	Portability
Μεταφραστής	Compiler
Μηχανική απαιτήσεων	Requirements engineering
Μονάδα	Unit
Μονάδα	Unit
Μοντέλο	Model
Μοντέλο διάταξης	Deployment model
Μοντέλο διάταξης ή εγκατάστασης	Deployment model
Μοντέλο λειτουργικής επαύξησης	Incremental model
Μοντέλο του πίδακα	Fountain model
Νέφος, υπολογιστικό νέφος	Cloud

Ολοκληρωμένα περιβάλλοντα προγραμματισμού	Integrated programming environments
Όνομα	Alias
Οντότητα	Entity
Οντοτήτων-συσχετίσεων	Entity-relationship
Πακέτο	Package
Παραγοντοποίηση	Factoring
Παρενέργεια	Side effect
Πεδίο	Field
Πελάτης	Client
Περιβάλλον εκτέλεσης λογισμικού	Runtime environment
Περιβάλλοντα γλωσσών προγραμματισμού	Programming language implementations
Περίπτωση ελέγχου	Test case
Περίπτωση χρήσης	Use case
Πηγαίος κώδικας	Source code
Πίνακας	Table
Πλεονάζων/ουσα	Redundant
Προδιαγραφή	Specification
Προσανατολισμένος-στα-αντικείμενα	Object-oriented
Προσανατολισμένος-στις-διαδικασίες	Function-oriented
Προστακτικός/ή	Imperative
Πρότυπο	Standard
Πρωτοτυποποίηση	Prototyping

Ρόλος	Role
Ρύθμιση	Calibration
Σtereότυπο	Stereotype
Συμβολομεταφραστής	Assembler
Συναλλαγή	Transaction
Συναρμολόγηση, συνάθροιση	Aggregation
Συνάρτηση	Function
Συναρτησιακή αποσύνθεση	Functional decomposition
Συναρτησιακός	Functional
Συνένωση	Integration
Συνοριακός	Boundary
Συνοριακός/ή	Boundary
Συντάκτης (κειμένου, προγράμματος)	(Text, program) editor
Συντήρηση λογισμικού	Software maintenance
Συστατικό στοιχείο λογισμικού	Component, artifact, module
Σύστημα τύπων	Type system
Συστήματα πραγματικού χρόνου	Real time systems
Συστήματα υποστήριξης λογισμικού	Software support system
Συσχέτιση	Relationship
Συσχέτιση	Association
Σχεδίαση	Design
Σχηματισμός	Configuration
Ταξινόμηση	Classification



Τεκμηρίωση	Documentation
Τεκμηρίωση λογισμικού	Software documentation
Τεχνολογία λογισμικού	Software engineering
Τράβηγμα	Drag
Τριμερής/πολυμερής διάταξη	3-tier, multi-tier
Τρόπος λειτουργίας	Mode
Τυπικός/ή	Formal
Υλικό	Hardware
Υπηρεσίες πάνω-από/μέσω web	Web services
Υπορουτίνα	Subroutine
Φυλομετρητής	Browser
Χειριστής	Actor
Χρωματική σύνταξη (προγράμματος)	Syntax highlighting
Institute of Electrical and Electronics Engineers	IEEE
Revision Control System	RCS
Source Code Control System	SCCS
Unified Modeling Language	UML

## ΓΝΩΡΙΜΙΑ ΜΕ ΤΗΝ ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ

### Σκοπός

---

Σκοπός του κεφαλαίου αυτού είναι ο ορισμός της έννοιας του λογισμικού, η τοποθέτησή του μέσα στις δραστηριότητες των οποίων η εκτέλεση πραγματοποιείται με τη βοήθειά του, ο εντοπισμός των κυριότερων κατηγοριών προβλημάτων που αφορούν την ανάπτυξη και συντήρησή του, καθώς και ο ορισμός της επιστημονικής περιοχής της Τεχνολογίας Λογισμικού.

### Προσδοκώμενα αποτελέσματα

---

Μετά τη μελέτη του κεφαλαίου αυτού, ο αναγνώστης θα είναι σε θέση:

- να αντιλαμβάνεται τις εφαρμογές λογισμικού ως τεχνικά κατασκευάσματα και την Τεχνολογία Λογισμικού ως την περιγραφή ενός πειθαρχημένου τρόπου για την κατασκευή τους,
- να περιγράφει τα σημαντικότερα προβλήματα στην ανάπτυξη του λογισμικού και τις αιτίες τους,
- να αναγνωρίζει τις διαφορετικές όψεις από τις οποίες μπορεί να ιδωθεί η ανάπτυξη του λογισμικού, καθώς και τη συσχέτιση αυτής με εξωγενείς παράγοντες.

### Έννοιες-κλειδιά

---

- Λογισμικό
- Τεχνικές κατασκευές
- Τεχνολογία Λογισμικού
- Ανάπτυξη λογισμικού
- Σύστημα
- Κρίση λογισμικού
- Συστατικό στοιχείο λογισμικού

Λίγοι θα διαφωνήσουν με τη θέση ότι το λογισμικό αποτέλεσε ένα από τα σημαντικότερα εργαλεία που ο άνθρωπος κατασκεύασε τον αιώνα που πέρασε, χάρη στο οποίο έγινε δυνατή η επανάσταση της πληροφορικής και των επικοινωνιών την οποία ακόμη βιώνουμε. Είναι αλήθεια ότι πολλοί άνθρωποι έχουν έρθει εκούσια ή ακούσια στη θέση του χρήστη μίας ή περισσότερων εφαρμογών λογισμικού, ωστόσο δεν είναι αυτονόητο ότι όλοι αντιλαμβάνονται το λογισμικό ως ένα μοναδικό τεχνικό κατασκεύασμα. Συνήθως, το λογισμικό γίνεται αντιληπτό ως εργαλείο, ως παιχνίδι ή γενικότερα ως μέρος ενός μεγαλύτερου και πιο σύνθετου συστήματος, ενώ σπάνια αντιλαμβανόμαστε το ίδιο το λογισμικό ως σύνθετο κατασκεύασμα με τα δικά του προβλήματα. Η εμπειρία δείχνει ότι η κατασκευή καλού λογισμικού δεν είναι καθόλου απλή υπόθεση. Συνήθως, πολλές παρανοήσεις χαρακτηρίζουν αυτό που γίνεται αντιληπτό ως κατασκευή λογισμικού, και ο αναγνώστης αναμφίβολα και αναπόφευκτα θα συναντήσει σημαντικό πλουραλισμό απόψεων κατά την ενασχόλησή του με το θέμα. Πέραν από τα καθαρά τεχνικά ζητήματα που αντιμετωπίζει ο κατασκευαστής του, πολλά από τα επιθυμητά χαρακτηριστικά του ίδιου του λογισμικού ή και της διαδικασίας κατασκευής του οφείλονται στην υπόστασή του ως προϊόντος. Αυτό καταδεικνύει την ανάγκη η κατασκευή του να ισορροπεί μεταξύ τεχνικής επάρκειας, μικρού κόστους και μεγάλης ταχύτητας. Με την ανάπτυξη των παραπάνω θεμάτων θα ασχοληθούμε στο κεφάλαιο αυτό.

## ΕΝΟΤΗΤΑ 1.1. ΥΠΟΛΟΓΙΣΤΕΣ ΚΑΙ ΛΟΓΙΣΜΙΚΟ

Ένα από τα σημαντικότερα γεγονότα που σηματοδότησαν τον αιώνα που πέρασε ήταν η εφεύρεση του ηλεκτρονικού υπολογιστή (Η/Υ). Με τη βοήθεια του ηλεκτρονικού υπολογιστή έγινε δυνατή η αυτοματοποίηση της εκτέλεσης πολλών κουραστικών, ανιαρών και επιρρεπών σε λάθη εργασιών, καθώς και η εκτέλεση άλλων, η οποία στο παρελθόν ήταν πρακτικά αδύνατη. Από την εποχή που κατασκευάστηκαν οι πρώτοι ηλεκτρονικοί υπολογιστές μέχρι σήμερα σημειώθηκε τεράστια βελτίωση των χαρακτηριστικών και των δυνατοτήτων τους. Κανένα άλλο ανθρώπινο κατασκεύασμα δεν σημείωσε τόσο σημαντική πρόοδο σε τόσο μικρό χρονικό διάστημα. Ένα από

τα πρακτικά αποτελέσματα αυτής της εξέλιξης ήταν ότι οι ηλεκτρονικοί υπολογιστές έγιναν προσιτοί σε μεγάλες μάζες ανθρώπων και αναφαίρετο εργαλείο της καθημερινής επαγγελματικής αλλά και ιδιωτικής ζωής για πολλούς από αυτούς. Παράλληλα, έγινε δυνατή η ενσωμάτωση των ηλεκτρονικών υπολογιστών σε πάρα πολλές συσκευές καθημερινής χρήσης, χωρίς αυτό να είναι πάντα αντιληπτό από τους ίδιους τους χρήστες. Σήμερα, σε πολλές από τις καθημερινές μας εργασίες χρησιμοποιούμε ηλεκτρονικούς υπολογιστές χωρίς να το γνωρίζουμε, ενώ συχνά η ίδια μας η ζωή εξαρτάται από αυτούς (υγεία, μέσα μεταφοράς, υπηρεσίες όπως έλεγχος οδικής και εναέριας κυκλοφορίας κ.ά.).

Η σημερινή εποχή μπορεί να χαρακτηριστεί ως μεταβατική σε μια νέα κατάσταση, στην οποία όλοι οι ηλεκτρονικοί υπολογιστές θα είναι διασυνδεδεμένοι μέσω δικτύων και θα εκτελούν σύνθετες εργασίες. Πολλές από τις σύγχρονες δικτυακές εφαρμογές μπορούν να μεταβάλουν κρίσιμες πλευρές του πολιτισμού μας, όπως την επικοινωνία, την εκπαίδευση και την κατάρτιση, αλλά και αυτή την ίδια τη λειτουργία των δημοκρατικών πολιτευμάτων. Η νέα κατάσταση που διαμορφώνεται αναφέρεται ως «κοινωνία της πληροφορίας» (information society) και έχουμε ήδη εισέλθει στο εξελικτικό της στάδιο με το διαδίκτυο και τις περί αυτού εφαρμογές να παίζουν πρωταγωνιστικό ρόλο στη διαδικασία. Όλες αυτές οι εξελίξεις γίνονται δυνατές χάρη στην ύπαρξη και λειτουργία ενός συνόλου πολύπλοκων εφαρμογών λογισμικού.

Η εξάπλωση του ηλεκτρονικού υπολογιστή σε ολοένα και περισσότερες πλευρές της ανθρώπινης ζωής δεν θα ήταν δυνατή χωρίς τη χρήση λογισμικού. Ο ηλεκτρονικός υπολογιστής ως συσκευή μπορεί μόνο να εκτελέσει ορισμένες πολύ απλές λειτουργίες με πάρα πολύ υψηλή ταχύτητα, όμως με τρόπο ιδιαίτερα δυσπρόσιτο στον άνθρωπο. Το λογισμικό είναι εκείνο που καθιστά χρήσιμη και αποδίδει στοιχεία «συμπεριφοράς» στη συσκευή του ηλεκτρονικού υπολογιστή. Ο άνθρωπος δεν αξιοποιεί τον ηλεκτρονικό υπολογιστή άμεσα ως συσκευή, αλλά μόνο μέσω του λογισμικού.

Έχοντας κατά νου τα παραπάνω, δεν είναι εύκολο να δοθεί ένας πλήρης και καθολικά αποδεκτός ορισμός της έννοιας «λογισμικό». Η πρακτική αξία αλλά και η διαχρονικότητα ενός θεωρητικού ορισμού μπορεί να αμφισβητηθεί σχετικά

εύκολα. Μια ικανοποιητική προσέγγιση είναι ο ορισμός του λογισμικού ως ακολούθως, τον οποίο και θα αποδεχτούμε ως επαρκή στο βιβλίο αυτό:

### Λογισμικό:

- (1) Προγράμματα ηλεκτρονικού υπολογιστή τα οποία όταν εκτελούνται επιτυγχάνουν επιθυμητά αποτελέσματα και επιδόσεις.
- (2) Δομές δεδομένων που επιτρέπουν σε προγράμματα να διαχειριστούν με επάρκεια κάθε λογής δεδομένα.
- (3) Κείμενα, διαγράμματα, μορφές παράστασης κ.ά., που περιγράφουν τη δομή, λειτουργία και χρήση των προγραμμάτων.

## ΕΝΟΤΗΤΑ 1.2. ΤΕΧΝΙΚΕΣ ΚΑΤΑΣΚΕΥΕΣ ΚΑΙ ΛΟΓΙΣΜΙΚΟ

Το λογισμικό είναι ένα πολύπλοκο τεχνικό κατασκεύασμα, το οποίο δεν έχει αυτοτελή υπόσταση, παρά μόνο όταν χρησιμοποιείται για να καθοδηγήσει έναν ηλεκτρονικό υπολογιστή στην πραγματοποίηση συγκεκριμένων λειτουργιών. Παρά τις αρκετές ομοιότητες που μπορεί κανείς συχνά να αναζητά μεταξύ λογισμικού και λοιπών τεχνικών κατασκευών, υπάρχουν και σημαντικές διαφορές. Η πρώτη είναι η μη απτή φύση του λογισμικού. Μια τεχνική κατασκευή είναι ορατή και απτή, ενώ το λογισμικό δεν είναι αυτό καθαυτό ορατό. Μόνο τα αποτελέσματα της χρήσης του μπορούν να είναι αντιληπτά. Η δομή του λογισμικού τόσο σε μικροσκοπικό όσο και σε μακροσκοπικό επίπεδο είναι και αυτή ένα νοητό κατασκεύασμα που μπορεί να γίνει αντιληπτό με διαφορετικούς τρόπους.

Η δεύτερη σημαντική διαφορά μπορεί να περιγραφεί από μία παρομοίωση: Σε αντίθεση με τα τεχνικά έργα, η κατασκευή των οποίων συνήθως ακολουθεί μια προκαθορισμένη οδό, γνωστή από την αρχή, η ανάπτυξη του λογισμικού ομοιάζει με *σκόπευση κινούμενου στόχου από κινούμενο έδαφος και με όπλο που συνεχώς αλλάζει τη συμπεριφορά του*. Ο στόχος είναι κινούμενος γιατί οι απαιτήσεις των χρηστών συνεχώς μεταβάλλονται, ακόμα και μέσα στη διαδικασία ανάπτυξης μιας εφαρμογής που προορίζεται να τις



ικανοποιήσει. Το έδαφος είναι κινούμενο γιατί το περιβάλλον ανάπτυξης του λογισμικού είναι και το ίδιο συνεχώς εξελισσόμενο μαζί με το υλικό αλλά και μαζί με τις επιλογές και την έκβαση των μη τεχνικών διαμαχών στο χώρο της αγοράς τεχνογνωσίας και τεχνολογίας πληροφορικής. Σαν να μην έφταναν τα παραπάνω, το όπλο με το οποίο γίνεται η σκόπευση, δηλαδή οι μεθοδολογίες, τα εργαλεία και τα περιβάλλοντα ανάπτυξης και λειτουργίας του λογισμικού είναι επίσης ραγδαία μεταβαλλόμενα με το χρόνο.

Όλοι μας γινόμαστε μάρτυρες ενός καταιγισμού προϊόντων, εξέλιξης λειτουργικών συστημάτων, γλωσσών προγραμματισμού, περιβαλλόντων και τεχνολογιών ανάπτυξης. Ο καταιγισμός αυτός φαίνεται να μην έχει ορατό τέλος, μιας και είναι οι νόμοι του ανταγωνισμού που σε πολλές περιπτώσεις προωθούν τις εξελίξεις αλλά και η ίδια η πρόοδος της τεχνολογίας των υπολογιστών η οποία είναι τουλάχιστον εντυπωσιακή.

Μολονότι η κατασκευή πολλών τεχνικών έργων είναι δυνατό να τυποποιηθεί σε αρκετά μεγάλο βαθμό, δεν ισχύει το ίδιο με την ανάπτυξη του λογισμικού. Η ανάπτυξη αυτή μέχρι σήμερα δεν έχει γίνει δυνατό να αυτοματοποιηθεί και το λογισμικό παραμένει ένα από τα πολυπλοκότερα και δυσκολότερα τεχνικά κατασκευάσματα του ανθρώπου, στην κατασκευή του οποίου συναντώνται επί μακράν σημαντικά προβλήματα, τα οποία από πολλούς χαρακτηρίζονται ως χρόνια.

## Δραστηριότητα I/Κεφάλαιο I

Εντοπίστε τουλάχιστον τρεις ομοιότητες και τρεις διαφορές μεταξύ του λογισμικού και της κατασκευής έργων οδοποιίας. Προσπαθήστε αυτές να καλύπτουν όσο το δυνατόν ευρύτερο φάσμα στοιχείων που σχετίζονται με την κατασκευή ή ανάπτυξη (κατασκευαστικά, διαχειριστικά, οικονομικά). Στο τέλος του κεφαλαίου μπορείτε να συγκρίνετε την απάντησή σας με τη δική μας προσέγγιση στο θέμα, η οποία προέκυψε μετά από πολυετή ενασχόληση τόσο με την ανάπτυξη εφαρμογών λογισμικού όσο και με τεχνικά έργα.



## ΕΝΟΤΗΤΑ 1.3. ΚΡΙΣΗ ΛΟΓΙΣΜΙΚΟΥ

Το σύνολο αυτών των χρόνιων προβλημάτων αναφέρεται ως «κρίση λογισμικού». Ενδεχομένως, η χρήση όρων όπως «κρίση» ή «χρόνια προβλήματα» μπορεί να χαρακτηριστεί υπερβολική, αυτό όμως δεν αναιρεί ούτε τη σοβαρότητα ούτε την παρατεταμένη διάρκεια εκδήλωσης των προβλημάτων που έχουν καταγραφεί και καθημερινά επιβεβαιώνονται στην ανάπτυξη του λογισμικού. Είναι χαρακτηριστικό ότι το λογισμικό είναι ένα από τα ελάχιστα ανθρώπινα κατασκευάσματα που πωλείται ως έχει, χωρίς *καμία* απολύτως εγγύηση για τις ζημιές που μπορεί να προκαλέσει η χρήση του, όσο σημαντικές και αν είναι αυτές. Ο Πίνακας I.I απεικονίζει τα σημαντικότερα από τα προβλήματα αυτά.

• Εξαιρετικά δύσκολη διαδικασία κατασκευής	Δεν είναι πάντα σαφές ποια βήματα πρέπει να γίνουν, με ποια σειρά, με ποια ενδιάμεσα προϊόντα κ.λπ.
• Ανεπαρκής ή και κακή ποιότητα τελικού προϊόντος	Λάθη στην κατασκευή, μη ικανοποίηση του σκοπού.
• Μη τήρηση χρονοδιαγραμμάτων	Υπερβολικές και αδικαιολόγητες καθυστερήσεις.
• Υπερβάσεις προϋπολογισμών	Κακές αρχικές εκτιμήσεις κόστους. Τελικά προϊόντα με πολλαπλάσιο κόστος από το αρχικά προϋπολογισθέν.
• Μεγάλη δυσκολία και συνεπαγόμενο κόστος συντήρησης	Παρενέργειες μεταβολών σε στοιχεία που πριν λειτουργούσαν, πρόχειρες λύσεις.
• Δύσκολη κατανόηση εγγράφων, σχεδίων κ.λπ. από διαφορετικούς κατασκευαστές	Στην πράξη, η κατανόηση ενός συστήματος λογισμικού από τρίτους, πλην των κατασκευαστών του, είναι συχνά αδύνατη ή ιδιαίτερα ασύμφορη.

**Πίνακας I.1** Μερικά βασικά σημεία της κρίσης λογισμικού.

Προβλήματα όπως τα παραπάνω έχουν εντοπιστεί εδώ και δεκαετίες από την κοινότητα κατασκευαστών και ακαδημαϊκών ερευνητών στη γνωστική περιοχή του λογισμικού και έχουν διατυπωθεί με πολλούς τρόπους και σε πολλές ευκαιρίες. Συχνά, νέες τεχνολογίες ή προϊόντα που προτείνονται για την ανάπτυξη του λογισμικού κάνουν επίκληση των προβλημάτων αυτών, ισχυριζόμενα ότι διαθέτουν ικανοποιητικές λύσεις. Για ένα διάστημα, οι λύσεις αυτές φέρονταν ως «ασημένια σφαίρα» που θα σκότωνε το «τέρας»

των προβλημάτων ανάπτυξης λογισμικού. Κάτι τέτοιο δεν έγινε και η φιλοσοφία της «ασημένιας σφαίρας» εγκαταλείφθηκε, για να πάρουν τη θέση της πιο συνετές επιστημονικές προσεγγίσεις στην ανάπτυξη του λογισμικού. Ως αποτέλεσμα, αναπτύχθηκε ένας ειδικός κλάδος της επιστήμης της πληροφορικής που ονομάστηκε «Τεχνολογία Λογισμικού» (Software Engineering). Πρόσφατα προτάθηκε η Τεχνολογία Λογισμικού να αποτελέσει εξειδίκευση της επιστήμης του μηχανικού, οπότε μια πιο εύστοχη απόδοση στα ελληνικά είναι αυτή της «Μηχανικής Λογισμικού». Στο βιβλίο αυτό θα αποδεχτούμε την απόδοση Τεχνολογία Λογισμικού ως επικρατέστερη για την ελληνική πραγματικότητα και θα αναφερόμαστε στην ίδια τεχνική επιστημονική περιοχή ανεξάρτητα από το αν πρόκειται για τεχνολογία, για μηχανική ή για... τέχνη, όπως συχνά υποστηρίζεται σε ακαδημαϊκές συζητήσεις.

## ΕΝΟΤΗΤΑ 1.4. ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ

Όπως ακριβώς με τον ορισμό της έννοιας «λογισμικό», προβλήματα και αρκετές διαφορετικές απόψεις υπάρχουν και στον ορισμό της «Τεχνολογίας Λογισμικού». Στο βιβλίο αυτό θα δεχτούμε τον παρακάτω ορισμό:

### **Τεχνολογία Λογισμικού:**

Η περιοχή εκείνη της επιστήμης της πληροφορικής η οποία ασχολείται με την εύρεση και θεμελίωση μεθόδων για να περιγράφεται, να κατασκευάζεται και να συντηρείται λογισμικό.

Επιθυμητά χαρακτηριστικά του λογισμικού και της διαδικασίας κατασκευής του είναι η ποιότητα, η μεγαλύτερη δυνατή αυτοματοποίηση και παραγωγικότητα και το ελάχιστο δυνατό κόστος παραγωγής και συντήρησης. Οι έννοιες ποιότητα, αυτοματοποίηση, παραγωγικότητα και κόστος είναι σε πολλές περιπτώσεις αντίθετες. Είναι φυσικό να μιλάμε όχι για ταυτόχρονη μεγιστοποίηση ποιότητας και παραγωγικότητας, από τη μία, και απόλυτη

ελαχιστοποίηση του κόστους, από την άλλη, αλλά για αποδεκτή –στις εκάστοτε συνθήκες– ισορροπία μεταξύ αυτών των μεγεθών.

Εντός του πεδίου της Τεχνολογίας Λογισμικού είναι ο καθορισμός των ενεργειών και της αλληλουχίας με την οποία αυτές πρέπει να γίνονται (software process), καθώς και η περιγραφή με σαφή και κατανοητό τρόπο όλων των προϊόντων που παράγονται κατά την εκτέλεση αυτών των ενεργειών. Το τελικό παραδοτέο προϊόν κάθε ενέργειας ανάπτυξης λογισμικού είναι ο εκτελέσιμος κώδικας, δηλαδή ένα σύνολο εντολών άμεσα εκτελέσιμων από έναν ηλεκτρονικό υπολογιστή κάτω από συγκεκριμένες (και γνωστές εκ των προτέρων) προϋποθέσεις. Το σύνολο αυτών των εντολών αποτελεί μια περιγραφή του τρόπου εκτέλεσης των εργασιών που αυτοματοποιούνται με τη χρήση μιας εφαρμογής λογισμικού.

Δεν είναι δυνατό η κατασκευή του λογισμικού να οδηγήσει κατευθείαν στον εκτελέσιμο κώδικα, όπως άλλωστε καμία απολύτως τεχνική κατασκευή δεν μπορεί να γίνει κατευθείαν, χωρίς να έχουν προηγηθεί μελέτες και σχέδια. Ωστόσο, ένα στοιχείο που διαφοροποιεί σημαντικά το λογισμικό από τις κλασσικές τεχνικές κατασκευές είναι ότι η κατασκευή του δεν είναι μια σειριακά ακολουθούμενη διαδικασία η οποία ολοκληρώνεται με την κατασκευή του παραδοτέου προϊόντος, αλλά το αρχικό αυτό παραδοτέο (πρώτη έκδοση εκτελέσιμου κώδικα και αντίστοιχο υλικό τεκμηρίωσης) υπόκειται συχνά σε πολλές τροποποιήσεις. Συνήθεις αιτίες για τροποποιήσεις στο λογισμικό είναι:

- η διόρθωση σφαλμάτων,
- η βελτιστοποίηση της απόδοσης,
- η αυτοματοποίηση της εκτέλεσης νέων εργασιών και
- η ενσωμάτωση μεταβολών που οφείλονται σε αλλαγές που συμβαίνουν στον πραγματικό κόσμο.

Η πραγματοποίηση μεταβολών/διορθώσεων στις εφαρμογές λογισμικού αναφέρεται με τον όρο «συντήρηση λογισμικού» (software maintenance). Όλες οι φάσεις από τις οποίες διέρχεται το λογισμικό αναφέρονται ως «κύκλος ζωής λογισμικού» (software life cycle). Γίνεται σαφές ότι **η Τεχνολογία Λογισμικού δεν ασχολείται μόνο με την κατασκευή, αλλά με ολόκληρο**

**τον κύκλο ζωής του λογισμικού.** Χρονικά, πρόκειται για το διάστημα από τη σύλληψη της ιδέας της κατασκευής μιας εφαρμογής λογισμικού μέχρι την απόσυρση αυτής από τη χρήση.

### **Δραστηριότητα 2/Κεφάλαιο I**

Αναφέρατε ένα παράδειγμα για καθεμία από τις αιτίες τροποποιήσεων στο λογισμικό οι οποίες αναφέρθηκαν στην προηγούμενη ενότητα. Στη συνέχεια, συγκρίνετε την απάντησή σας με αυτή που δίνουμε στο τέλος του κεφαλαίου. Σε πόσες περιπτώσεις συμπίπτουν οι απαντήσεις;

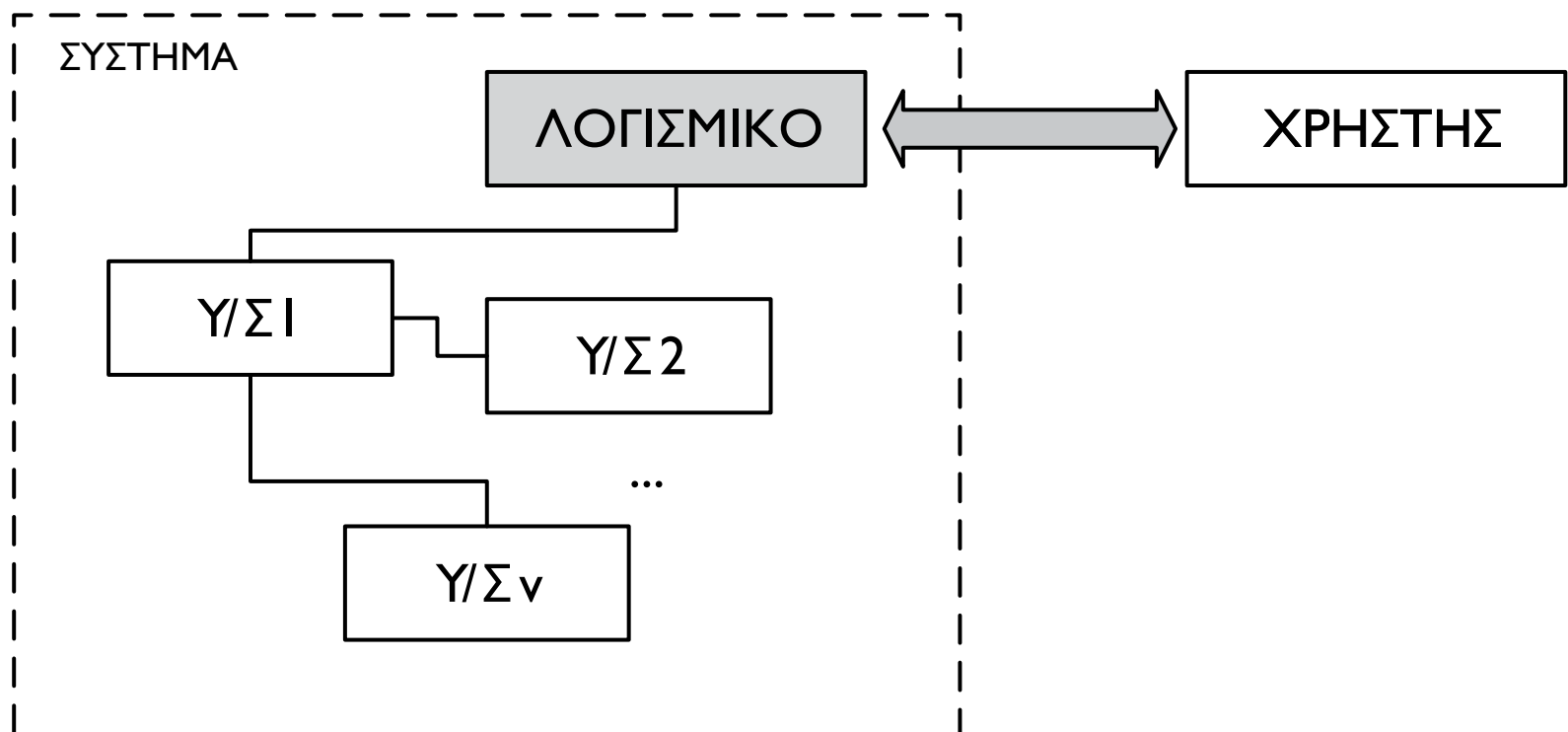
## **ΕΝΟΤΗΤΑ 1.5. ΤΟ ΛΟΓΙΣΜΙΚΟ ΩΣ ΜΕΡΟΣ ΣΥΣΤΗΜΑΤΩΝ**

Συχνά, το λογισμικό αντιμετωπίζεται λανθασμένα όχι ως μέρος ενός ευρύτερου συστήματος με το οποίο αλληλεπιδρά με πολλούς τρόπους αλλά ως αυθύπαρκτη οντότητα. Στην Τεχνολογία Λογισμικού συχνά γίνεται ξεχωριστά λόγος για το *σύστημα* και ξεχωριστά για το *λογισμικό*, και δεν είναι λίγες οι περιπτώσεις όπου μπορεί να δημιουργηθεί σύγχυση σχετικά με τις έννοιες και την προσέγγιση πολλών οντοτήτων του πραγματικού κόσμου κατά την ανάπτυξη λογισμικού. Θα διακρίνουμε δύο περιπτώσεις:

- Το λογισμικό αποτελεί εσωτερικό συστατικό ενός τεχνητού, μη υπολογιστικού συστήματος.
- Το λογισμικό λειτουργεί αυτοτελώς σε ένα υπολογιστικό σύστημα.

Ως παραδείγματα για το πρώτο μπορούμε να αναφέρουμε όλες τις περιπτώσεις όπου μια συσκευή λειτουργεί χρησιμοποιώντας λογισμικό, όπως οι μηχανές αυτόματης πώλησης, οι ψηφιακοί αυτοματισμοί και, σύντομα, αρκετές οικιακές συσκευές. Επίσης, στην πρώτη κατηγορία ανήκουν σύνθετα συστήματα όπου το λογισμικό ή η υπολογιστική μονάδα στην οποία αυτό εκτελείται λειτουργεί συνδεδεμένη με άλλες συσκευές, όπως τα συστήματα χρονομέτρησης αγώνων, τα ιατρικά μηχανήματα ανάλυσης και απεικόνισης, τα συστήματα ελέγχου εναέριας κυκλοφορίας κ.ά. (Σχήμα I.I).

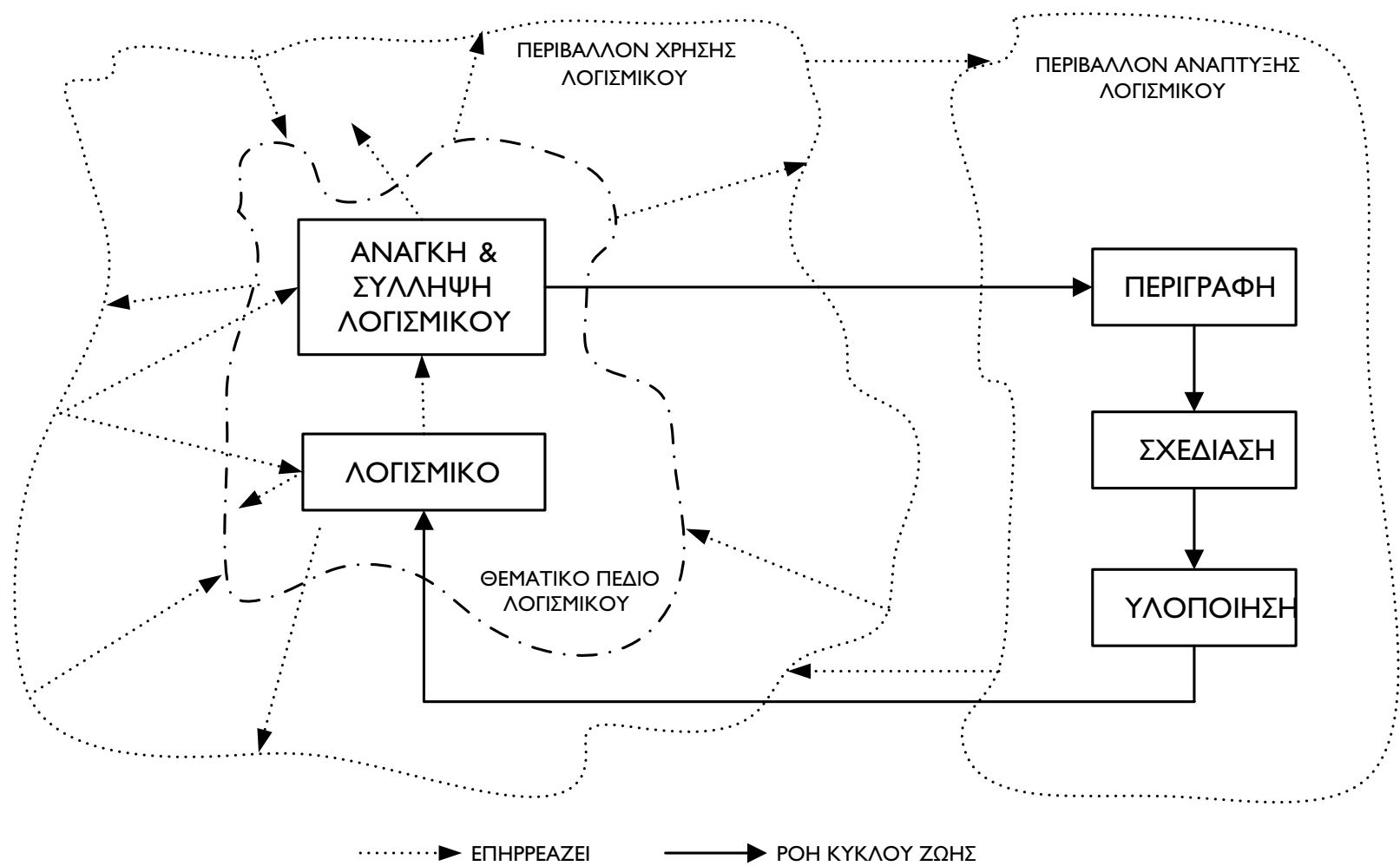
**Σχήμα I.1** Το λογισμικό μπορεί να είναι μέρος πολλών συστημάτων. Ο χρήστης, αλληλεπιδρώντας με τα συστήματα, μπορεί να χρησιμοποιεί λογισμικό χωρίς να έχει άμεση αντίληψη του γεγονότος αυτού.



Στις περιπτώσεις αυτές, κατά την ανάπτυξη του λογισμικού οφείλουν να λαμβάνονται υπόψη τα ειδικά χαρακτηριστικά των συσκευών που αποτελούν τα υπόλοιπα μέρη του συστήματος. Τα χαρακτηριστικά τέτοιων συσκευών, χωρίς να αποτελούν αυτά καθαυτά χαρακτηριστικά του λογισμικού, καθορίζουν σε μεγάλο βαθμό τη δομή και τη συμπεριφορά του.

Στη δεύτερη περίπτωση, όπου το λογισμικό στεγάζεται απλώς σε ένα υπολογιστικό σύστημα, δεν αποτελεί με τη δομική αλλά με τη λειτουργική έννοια μέρος ενός ευρύτερου οργανισμού από τον οποίο καθορίζεται και τον οποίο με τη σειρά του καθορίζει (Σχήμα Ι.2). Από την έναρξη μέχρι την ολοκλήρωση της ανάπτυξης μιας εφαρμογής λογισμικού λαμβάνουν συνήθως χώρα αρκετές αλλαγές στο ιδιαίτερο και το ευρύτερο πεδίο χρήσης αυτής, στις οποίες ενίοτε αποδίδεται μέρος των αιτιών αποτυχίας ή αστοχίας της προσπάθειας ανάπτυξης.

**Σχήμα I.2** Αλληλεπιδράσεις στην ανάπτυξη του λογισμικού.





Μπορεί κανείς να αναγνωρίσει μια διαλεκτική σχέση μεταξύ του λογισμικού και του χώρου στον οποίο αυτό αναπτύσσεται και χρησιμοποιείται: ο τρόπος με τον οποίο γίνονται οι εργασίες στον πραγματικό κόσμο επιβάλλει την αυτοματοποίηση ορισμένων από αυτές, η οποία γεννάει την ανάγκη για λογισμικό. Η ενσωμάτωση της αυτοματοποίησης στο χώρο που γέννησε την ανάγκη επιδρά εκ νέου στη σύλληψη των εργασιών και τη σχεδίαση του τρόπου με τον οποίο αυτές γίνονται, πράγμα που με τη σειρά του μπορεί να επιφέρει μεταβολές στο λογισμικό και ο κύκλος επαναλαμβάνεται. Πέραν αυτής της άμεσης αλληλεπίδρασης μεταξύ πεδίου εφαρμογής και λογισμικού, είναι και ο ίδιος ο κόσμος που μεταβάλλεται συνεχώς από εσωτερικές δυνάμεις, πράγμα που διαμορφώνει, εξελίσσει ή καταργεί ολοένα και περισσότερες εργασίες οι οποίες μπορούν να αυτοματοποιηθούν με τη χρήση λογισμικού.

### **Δραστηριότητα 3/Κεφάλαιο I**

Αναφέρατε τουλάχιστον τρία παραδείγματα διατάξεων (συσκευών) που δεν είναι ηλεκτρονικοί υπολογιστές και ενσωματώνουν λογισμικό με το οποίο επικοινωνεί με κάποιο τρόπο ο χρήστης. Αν σκεφτεί κανείς πόσες τέτοιες συσκευές υπάρχουν και πόσο συχνή είναι η χρήση τους, το πιθανότερο είναι ότι θα εντυπωσιαστεί από τη διάδοση του λογισμικού. Είμαστε σχεδόν σίγουροι ότι θα εμπλουτίσετε σημαντικά την απάντηση που δίνουμε στο τέλος του κεφαλαίου.

## **ΕΝΟΤΗΤΑ 1.6. ΤΟ ΛΟΓΙΣΜΙΚΟ ΩΣ ΠΡΟΪΟΝ**

Οι περισσότερες προσπάθειες ανάπτυξης λογισμικού στοχεύουν στη δημιουργία ενός προϊόντος, δηλαδή ενός αγαθού το οποίο προορίζεται να βρει το δρόμο του στην αγορά είτε ως προϊόν μαζικής κατανάλωσης είτε κατασκευασμένο κατά παραγγελία. Η περίπτωση στην οποία μπορεί κανείς να κατασκευάσει λογισμικό για δική του προσωπική χρήση δεν διαφέρει

σε τίποτε από τεχνικής άποψης από την κατασκευή λογισμικού για μαζική κατανάλωση, είναι ωστόσο περιορισμένου ενδιαφέροντος, με το σκεπτικό ότι αποτελεί μια εξαίρεση στην οποία όποιοι κανονισμοί επιβάλλονται από επιστημονικά θεμελιωμένες προσεγγίσεις μπορούν να τηρηθούν με μεγάλη χαλαρότητα.

Σε αντίθεση με τις θεωρητικές επιστήμες όπου η δυνατότητα χρήσης απεριόριστου χρόνου προκειμένου να επιτευχθεί ένας στόχος είναι μεγαλύτερη, οι νόμοι του ανταγωνισμού και η πίεση του χρόνου δεν επιτρέπουν κάτι τέτοιο με το λογισμικό. Η επιδίωξη της κατάκτησης μιας καλής θέσης στην αγορά περιορίζει τη δυνατότητα εφαρμογής εξαντλητικών διαδικασιών εξασφάλισης ποιότητας και αποτελεί μία από τις βασικές αιτίες για πολλές από τις πληγές του λογισμικού σήμερα. Η λειτουργία των νόμων της αγοράς έχει και άλλο ένα ενδιαφέρον αποτέλεσμα: το καλύτερο από τεχνικής πλευράς λογισμικό δεν είναι κατ' ανάγκη και το επικρατέστερο στην αγορά. Σε αρκετές περιπτώσεις, διαμάχες για τεχνολογίες και προϊόντα λογισμικού, οι οποίες εμφανίζονται ως τεχνικές διαμάχες, μόνο τέτοιες δεν είναι. Ο ανταγωνισμός δυσχεραίνει τη συνεργασία με σκοπό την αναζήτηση του καλύτερου δυνατού αποτελέσματος. Συχνά, περισσότερες από μία πλευρές κατέχουν η καθεμία ένα μέρος της επιθυμητής λύσης σε ένα πρόβλημα χωρίς καμία να κατέχει ολόκληρη τη λύση και χωρίς να βαδίζουν από κοινού σε δρόμο σύγκλισης. Κάποιες φορές, οι διαμάχες αυτές αποβαίνουν σε όφελος του τελικού καταναλωτή, ενώ σε πολλές περιπτώσεις συμβαίνει ακριβώς το αντίθετο.

Αν επιχειρήσουμε μια γενική ταξινόμηση των προϊόντων λογισμικού, μπορούμε να διακρίνουμε δύο γενικές κατηγορίες. Το **λογισμικό συστήματος** και το **λογισμικό εφαρμογών**. Ως **λογισμικό συστήματος** γίνεται αντιληπτό εκείνο το λογισμικό χωρίς το οποίο δεν είναι δυνατή η λειτουργία ενός ηλεκτρονικού υπολογιστή, δηλαδή τα λειτουργικά συστήματα (operating systems). Οι υποκατηγορίες που μπορούν να εντοπιστούν είναι τα λειτουργικά συστήματα γενικής χρήσης (Unix, DOS, Windows), καθώς και ειδικές περιπτώσεις, όπως λογισμικό προγραμματισμού (όχι εφαρμογές) αυτόματων ελεγκτών στη βιομηχανία.

Στην κατηγορία του **λογισμικού εφαρμογών** ανήκουν γενικά όλες οι υπολοίπες περιπτώσεις. Και εδώ μπορούμε να διακρίνουμε υποκατηγορίες, όπως το λογισμικό επιχειρηματικών εφαρμογών, εφαρμογών πραγματικού χρόνου, επιστημονικών εφαρμογών, εκπαιδευτικών εφαρμογών, προσωπικής χρήσης, τεχνητής νοημοσύνης κ.ά. Μπορούν να αναζητηθούν και άλλες κατηγορίες ή υποκατηγορίες σύμφωνα με άλλη κεντρική ιδέα ταξινόμησης, ώστε ο κατάλογος να συνεχίζεται και να εξειδικεύεται επί μακρόν. Ο ρυθμός των εξελίξεων και η χρήση του λογισμικού σε ολόένα και περισσότερες πλευρές της καθημερινής ζωής, όχι μόνο στις επαγγελματικές, έχει καταστήσει ανεπίκαιρες όλες τις απόπειρες ολοκληρωμένης ταξινόμησης του λογισμικού σε κατηγορίες, οι οποίες έχουν γίνει στο παρελθόν.

**Η υπόσταση του λογισμικού ως προϊόντος επιβάλλει για την Τεχνολογία Λογισμικού τη διατύπωση ενός συνόλου κανόνων και διαδικασιών ανάπτυξης που να ισορροπούν μεταξύ τεχνικής ορθότητας (στο μέτρο που αυτή είναι θεμελιωμένη) από τη μία και οικονομικής εφικτότητας από την άλλη. Επιπλέον, η ανάπτυξη λογισμικού οφείλει να γίνεται σε λογικό χρόνο, ώστε αυτό να εισέρχεται στην αγορά σε στιγμή που η ζήτηση είναι υψηλή και να αποφέρει κέρδη στον κατασκευαστή του. Εν συντομία, η Τεχνολογία Λογισμικού, ιδωμένη είτε ως μηχανική, είτε ως επιστήμη, είτε ως τέχνη είναι μια πρόκληση για όσους αποφασίσουν να ασχοληθούν με αυτή είτε ερευνητικά είτε στο πεδίο της «μαχόμενης πληροφορικής».**

#### **Δραστηριότητα 4/Κεφάλαιο I**

Αναφέρατε τουλάχιστον τρία παραδείγματα λογισμικού συστήματος και πέντε παραδείγματα λογισμικού εφαρμογών τα οποία να είναι προϊόντα που βρίσκονται στην αγορά. Δεν έχετε παρά να κοιτάξετε μια βιτρίνα καταστήματος υψηλής τεχνολογίας ή να κάνετε μια βόλτα σε δικτυακούς τόπους. Μερικά χαρακτηριστικά αποτελέσματα από τη δική μας βόλτα παρατίθενται στο τέλος του κεφαλαίου.

## ΕΝΟΤΗΤΑ 1.7. ΣΥΣΤΑΤΙΚΑ ΣΤΟΙΧΕΙΑ ΛΟΓΙΣΜΙΚΟΥ

Σύμφωνα με τον ορισμό που δόθηκε στην ενότητα Ενότητα I.I, λογισμικό δεν είναι μόνο ένα εκτελέσιμο πρόγραμμα. Οι χρήστες συνήθως αντιλαμβάνονται ως λογισμικό το πρόγραμμα μαζί με το αντίστοιχο εγχειρίδιο χρήσης. Πέραν αυτών, μέρος του λογισμικού είναι και πολλά ενδιάμεσα προϊόντα τα οποία παράγονται στις φάσεις που μεσολαβούν από τον καθορισμό των εργασιών που θα αυτοματοποιηθούν με τη βοήθεια του λογισμικού μέχρι την παραγωγή του εκτελέσιμου κώδικα. Τα προϊόντα αυτά είτε είναι ενδιάμεσα συστατικά λογισμικού που παράγονται μέχρι να φτάσουμε στον εκτελέσιμο κώδικα (πηγαίος κώδικας, κώδικας μορφής object, βιβλιοθήκες κ.ά.) είτε περιγράφουν τη δομή και τη συμπεριφορά του λογισμικού. Στη δεύτερη περίπτωση αναφέρονται με τον όρο «τεκμηρίωση λογισμικού» (software documentation) και βρίσκονται σε έντυπη ή σε ηλεκτρονική μορφή. Κατά παρέκκλιση της αυστηρής λεξικογραφικής σημασίας της λέξης, σε αυτή την τεκμηρίωση του λογισμικού δεν καταγράφεται το γιατί το λογισμικό εκτελεί κάποιες εργασίες ή γιατί τις εκτελεί με ένα συγκεκριμένο τρόπο, αλλά το ποιες εργασίες θα εκτελεί, πώς θα τις εκτελεί, ποιες δομές δεδομένων θα χρησιμοποιηθούν κ.ά.

### **Συστατικά λογισμικού:**

Είναι όλα τα προϊόντα που παράγονται κατά την ανάπτυξη του λογισμικού, τα οποία αποτελούν αναπόσπαστο μέρος αυτού.

Τα συστατικά του λογισμικού μπορούν να ταξινομηθούν ως προς τη φύση τους, τον τρόπο παραγωγής τους, τη φάση του κύκλου ζωής στην οποία παράγονται, την εσωτερική τους δομή, τα πρότυπα στα οποία ενδεχομένως συμμορφώνονται κ.ά. Ως προς τη φύση διακρίνουμε αυτά που βρίσκονται σε ηλεκτρονική μορφή και αυτά που βρίσκονται σε έντυπη. Ως προς τον τρόπο παραγωγής τους διακρίνουμε αυτά που παράγονται αυτόματα (κώδικας μορφής object, εκτελέσιμος κώδικας, περιγραφή σχημάτων βάσεων δεδομένων κ.ά.) και αυτά που παράγονται με το χέρι. Η ταξινόμηση ως προς την εσωτερική τους δομή ποικίλει ανάλογα με την τεχνική φύση του

περιβάλλοντος ανάπτυξης και λειτουργίας. Τέλος, η συμμόρφωση με πρότυπα αναφέρεται στη δομημένη περιγραφή ορισμένων χαρακτηριστικών του λογισμικού, με το «δομημένη» να αφορά την πειθαρχία απέναντι σε όσα ορίζονται σε ένα ή περισσότερα πρότυπα (standards) που χρησιμοποιούνται για το σκοπό αυτό.

Η αναφορά στα πρότυπα μας φέρνει σε ένα σημαντικό πρόβλημα στο χώρο του λογισμικού. Σήμερα, μέσα στην κοινότητα κατασκευαστών και ερευνητών του λογισμικού υπάρχει ένας πλουραλισμός συμβόλων, τίτλων, ορισμών εννοιών, δομών κ.λπ., που αναφέρονται σε παρεμφερείς οντότητες, χωρίς να διευκολύνουν μια καθολική, σαφή και χωρίς διφορούμενα κατανόηση των οντοτήτων που σχετίζονται με το λογισμικό. Ο πλουραλισμός αυτός είναι έκδηλος παντού: στις μεθοδολογίες ανάπτυξης, στις γλώσσες προγραμματισμού, στα εργαλεία και αλλού, και οδηγεί τους κατασκευαστές στην επιλογή δικών τους επιλύσεων σε διφορούμενα θέματα τεκμηρίωσης και στη χρήση δικών τους άτυπων συμβολισμών και δομών, γεγονός που μάλλον δυσχεραίνει το πρόβλημα και αυξάνει τη σύγχυση.

Αιτίες της σύγχυσης αυτής μπορούν να αναγνωριστούν σε αρκετά επίπεδα. Πέραν από την αρχική ανωριμότητα η οποία χαρακτηρίζει κάθε νέο ερευνητικό πεδίο, υπάρχει ο ανταγωνισμός για την εμπορική επικράτηση σε τομείς όπως τα εργαλεία ανάπτυξης και η παροχή τεχνογνωσίας για την ανάπτυξη λογισμικού. Ακόμα και οι συμβολισμοί και η ορολογία αποτελούν πεδίο διαμάχης και σύγχυσης. Μια αξιοσημείωτη προσπάθεια για την ανάπτυξη προτύπων για την περιγραφή πολλών συστατικών λογισμικού έκανε ο οργανισμός IEEE (Institute of Electrical and Electronics Engineers). Τα πρότυπα αυτά προσαρμόζονται ανάλογα με τη μεθοδολογία ανάπτυξης και τον ακολουθούμενο κύκλο ζωής. Ωστόσο, έχουν ένα κάθε άλλο παρά αμελητέο κόστος συγγραφής και, ιδιαίτερα, διατήρησής τους σε επίκαιρη κατάσταση, με αποτέλεσμα σε αρκετές περιπτώσεις είτε να καταργούνται στην πράξη είτε να μένουν χωρίς να ενημερώνονται σχετικά με τις μεταβολές που λαμβάνουν χώρα από την αρχική συγγραφή τους και μετά.



## Άσκηση I/Κεφάλαιο I

**Ποια από τα παρακάτω είναι συστατικά λογισμικού και ποια όχι;**

1. Έκθεση αναγκών του πελάτη
2. Ενημερωτικό έντυπο για κάποια εφαρμογή
3. Σχέδιο δομής λογισμικού
4. Γλώσσα προγραμματισμού
5. Μηνύματα ασφαμάτων μεταγλώττισης
6. Περιγραφή των λειτουργιών του λογισμικού
7. Έκθεση προβλημάτων πελάτη
8. Περιγραφή ενεργειών ελέγχου λογισμικού
9. Εκτύπωση αποτελεσμάτων μιας εφαρμογής λογισμικού

Δείτε την απάντηση που δίνεται στο τέλος του κεφαλαίου και συγκρίνετέ τη με τη δική σας.

## ΕΝΟΤΗΤΑ 1.8.

### Σύνοψη

---

Το λογισμικό είναι ένα σύνθετο τεχνικό κατασκεύασμα που προορίζεται στο να συμβάλλει στην αυτοματοποίηση επίπινων και επιρρεπών σε σφάλματα ανθρώπινων εργασιών με τη βοήθεια ηλεκτρονικού υπολογιστή. Ως λογισμικό δεν εννοείται μόνο ο εκτελέσιμος κώδικας αλλά και ένα σύνολο ενδιάμεσων προϊόντων, όπως προδιαγραφές, σχέδια, πηγαίος κώδικας, εκθέσεις ελέγχου κ.ά. Όλα αυτά αποτελούν παράγωγα προϊόντα του κύκλου ζωής του λογισμικού, ο οποίος περιλαμβάνει όλες τις φάσεις, από τη σύλληψη της ιδέας μέχρι και την απόσυρση μιας εφαρμογής λογισμικού από τη χρήση. Παρά τη σημαντική πρόοδο που έχει επιτευχθεί στον τομέα του υλικού των υπολογιστών, η κατασκευή του λογισμικού παρουσιάζει ορισμένα χρόνια, σημαντικά προβλήματα που σχετίζονται με την ποιότητα, το κόστος και τη γενική επάρκεια του τρόπου με τον οποίο αυτή γίνεται. Τα προβλήματα αυτά αναφέρονται γενικά ως «κρίση λογισμικού» (software crisis). Η Τεχνολογία Λογισμικού είναι η περιοχή εκείνη της επιστήμης της μηχανικής που ασχολείται με την εύρεση και θεμελίωση μεθόδων για να περιγράφεται, να κατασκευάζεται και να συντηρείται λογισμικό καλής ποιότητας με τη μεγαλύτερη δυνατή αυτοματοποίηση και παραγωγικότητα και το ελάχιστο δυνατό κόστος. Η Τεχνολογία Λογισμικού δεν είναι μια θεωρητική επιστήμη αλλά στοχεύει στην υποστήριξη των κατασκευαστών να παράγουν καλά προϊόντα λογισμικού. Τα προϊόντα αυτά αντιμετωπίζονται ως αναπόσπαστα τμήματα του ειδικότερου και ευρύτερου πεδίου χρήσης αυτών, από το οποίο επηρεάζονται και το οποίο επηρεάζουν.

## ΕΝΟΤΗΤΑ 1.9. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ

### Δραστηριότητα I/Κεφάλαιο I

---

#### Ομοιότητες

1. Τόσο οι εφαρμογές λογισμικού όσο και τα έργα οδοποιίας πρέπει να κατασκευάζονται με συγκεκριμένο περιοριστικό χρονοδιάγραμμα και προϋπολογισμό.
2. Η κατασκευή αμφοτέρων πρέπει να γίνει σε διακριτές φάσεις οι οποίες σχετίζονται μεταξύ τους.
3. Δεν είναι δυνατή η κατασκευή ούτε του λογισμικού ούτε των έργων οδοποιίας χωρίς τη χρήση ενός λεπτομερούς σχεδίου.
4. Η διάταξη και η αλληλοσύνδεση των επιμέρους στοιχείων τόσο του λογισμικού όσο και των συστατικών ενός έργου οδοποιίας (υπόβαθρο, ασφαλτος, συστήματα ενημέρωσης οδικής κυκλοφορίας) επηρεάζουν τη συνολική συμπεριφορά του και την εικόνα που αυτό εμφανίζει στους χρήστες του.
5. Οι απαιτήσεις από αμφότερα τα κατασκευάσματα τίθενται από τους χρήστες αυτών. Στην περίπτωση του λογισμικού μιλάμε για τον πελάτη, ο οποίος παραγγέλλει το λογισμικό, ενώ στα έργα οδοποιίας χρήστες είναι το κοινωνικό σύνολο, τις απαιτήσεις του οποίου περιγράφουν οι εκπρόσωποι αυτού.

#### Διαφορές

1. Αρκετά από τα ποιοτικά χαρακτηριστικά ενός έργου οδοποιίας είναι ορατά με γυμνό μάτι, πράγμα που δεν ισχύει για το λογισμικό.
2. Η ευστάθεια και η διάρκεια στο χρόνο ενός έργου οδοποιίας απειλείται περισσότερο από φυσικά φαινόμενα και σχεδόν καθόλου από τη φυσιολογική χρήση, πράγμα που δεν ισχύει για το λογισμικό.
3. Η πιθανότητα κατά την κατασκευή ενός έργου οδοποιίας να αλλάξουν οι απαιτήσεις του πελάτη, ώστε αυτή να πρέπει να ξεκινήσει



από μηδενική βάση, είναι πολύ μικρότερη από την αντίστοιχη πιθανότητα για την κατασκευή μιας εφαρμογής λογισμικού.

4. Τα σχέδια των έργων οδοποιίας είναι σαφή και ακολουθούν ένα καθολικά αποδεκτό πρότυπο, πράγμα που δεν ισχύει για τα σχέδια του λογισμικού.

Ο αναγνώστης προτρέπεται να αναγνωρίσει και άλλες διαφορές, εστιάζοντας την προσοχή του κυρίως στη μη απτή φύση του λογισμικού αλλά και στη συχνά παρατηρούμενη ραγδαία μεταβολή των αρχικών απαιτήσεων του πελάτη πριν ακόμη ολοκληρωθεί η κατασκευή του λογισμικού που τις ικανοποιεί.

Πάντως, αν δεν εντοπίσατε αρκετές ομοιότητες ή διαφορές, αυτό οφείλεται προφανώς στο νέο της γνωριμίας σας με το λογισμικό και πιθανόν και με τα τεχνικά έργα. Όταν θα έχετε προχωρήσει στη μελέτη του βιβλίου αυτού και, παράλληλα, εργαζόμενοι σε ασκήσεις της θεματικής ενότητας, θα είστε σε θέση να εντοπίζετε ολοένα και περισσότερες τέτοιες ομοιότητες και διαφορές

## Δραστηριότητα 2/Κεφάλαιο I

Αιτία	Παράδειγμα
Διόρθωση σφαλμάτων	Το πρόγραμμα αρνείται να λειτουργήσει κάτω από συγκεκριμένες συνθήκες και παράγει ένα λάθος κατά την εκτέλεση.
Βελτιστοποίηση της απόδοσης	Το πρόγραμμα παράγει τα επιθυμητά αποτελέσματα σε χρόνο που μπορεί να βελτιωθεί (για τον ίδιο υπολογιστή).
Αυτοματοποίηση νέων εργασιών	Διαπιστώνεται ότι υπάρχει μια εργασία η οποία θα ήταν χρήσιμο να προστεθεί σε αυτές που εκτελεί το πρόγραμμα.
Ενσωμάτωση μεταβολών από τον πραγματικό κόσμο	Αλλάζει το νομικό πλαίσιο της τήρησης «βιβλίων και στοιχείων» και οι αλλαγές αυτές πρέπει να ενσωματωθούν στο λογισμικό

*Σημείωση: Για ευνόητους λόγους δεν γίνονται αναφορές σε συγκεκριμένες εφαρμογές λογισμικού. Ωστόσο, η διαπίστωση ότι το καλό μπορεί πάντα να γίνει καλύτερο μπορεί να μας οδηγήσει στην εύρεση και άλλων τέτοιων παραδειγμάτων, ιδιαίτερα σε έναν κόσμο όπως ο δικός μας, στον οποίο όλα αλλάζουν με γοργούς ρυθμούς.*

## Δραστηριότητα 3/Κεφάλαιο I

- Πωλητής αναψυκτικών
- Αυτόματο ανταλλακτήριο συναλλάγματος
- Αυτόματος κλιματισμός χώρου
- Σύστημα χρονισμού και ελέγχου κινητήρα αυτοκινήτου
- Σύστημα ελέγχου ανελκυστήρα
- Συσκευή video, audio CD, ψηφιακά ηχητικά μηχανήματα κ.λπ.

- Κινητό τηλέφωνο

Μπορείτε να ανακαλύψετε πολλά ακόμη τέτοια παραδείγματα. Σκεφτείτε ότι οποιοσδήποτε μικρός ή μεγάλος αυτοματισμός στις σύγχρονες συσκευές υλοποιείται με τη βοήθεια λογισμικού. Αυτό ισχύει και στις πλέον απίθανες περιπτώσεις (π.χ. ηλεκτρική κουζίνα ή ηλεκτρικό κάθισμα αυτοκινήτου με μνήμες).

## Δραστηριότητα 4/Κεφάλαιο Ι

---

Συστήματος	Εφαρμογών
Apple OSX	Oracle
Microsoft Windows 7	SAP
Google Android	Adobe Illustrator
Microsoft Windows 10	Autocad
Linux	Microsoft Word

Σε επίπεδο λειτουργικών συστημάτων θα βρείτε αρκετά ακόμη αν ανατρέξετε σε μια μηχανή αναζήτησης του διαδικτύου (π.χ. [www.metacrawler.com](http://www.metacrawler.com)) ζητώντας πληροφορίες για computer operating systems. Σε επίπεδο λογισμικού εφαρμογών μπορείτε, όπως είπαμε, να κάνετε μια βόλτα σε κάποιο κατάστημα προϊόντων υψηλής τεχνολογίας και να δείτε εκατοντάδες ακόμη τίτλους.

## Άσκηση I/Κεφάλαιο I

---

Συστατικά λογισμικού είναι τα 1, 3, 6, 8.

Από τα υπόλοιπα, έχει νόημα να αναφερθούν παρατηρήσεις για τα 4, 5 και 7: το (4) είναι εργαλείο ανάπτυξης, το (5) είναι παράγωγο του εργαλείου, το (7) δεν είναι το ίδιο (αν και μπορεί να πυροδοτήσει τη διαδικασία δημιουργίας συστατικών λογισμικού).

Δεν είναι εύκολη η διάκριση των συστατικών λογισμικού από την αρχή. Ακόμα και πεπειραμένοι μηχανικοί λογισμικού συχνά υποτιμούν κάτι και δεν το κατατάσσουν στα συστατικά λογισμικού. Όσο προχωράτε στη μελέτη της ύλης του βιβλίου αυτού θα σας είναι περισσότερο προφανής η αντίληψη των συστατικών λογισμικού, τα οποία απλά θα αποκαλύπτονται μπροστά σας αβίαστα. Για την ώρα, αν δεν μείνατε ικανοποιημένοι από την απάντηση που δώσατε, μπορείτε να ανατρέξετε στην Ενότητα I.1 καθώς και στην Ενότητα I.7.

## ΒΙΒΛΙΟΓΡΑΦΙΑ

Agresti, W. W., *New Paradigms for Software Development*, IEEE Computer Society Order Number 707, IEEE Computer Society Press, 1986.

Brooks, F. P. (1982) *The Mythical Man Month*, Reading, Massachusetts: Addison-Wesley.

Budgen, D., *Introduction to Software Engineering*, SEI Curriculum Module SEI-CM-2-2.1, Software Engineering Institute, Carnegie Mellon University, 1989.

Fairley, R. E., *Software Engineering Concepts*, McGraw-Hill, 1985.

*IEEE Standard Glossary of Software Engineering Terminology*, ANSI/IEEE, Std 610.12-1990

Jackson, M.A., *System Development*, Englewood Cliffs, N. J.: Prentice-Hall, 1983.

Macro, A. and J. Buxton, *The Craft of Software Engineering*, Addison-Wesley, 1987.

Pressman, R. S., *Software Engineering-A Practitioners Approach*, McGraw-Hill.

Sommerville, I. *Software Engineering*, London: Addison-Wesley.

## ΜΟΝΤΕΛΑ ΚΥΚΛΟΥ ΖΩΗΣ ΛΟΓΙΣΜΙΚΟΥ

### Σκοπός

---

Σκοπός του κεφαλαίου είναι ο ορισμός της έννοιας του μοντέλου κύκλου ζωής λογισμικού, η γνωριμία με τα πιο διαδεδομένα από αυτά τα μοντέλα, η ταξινόμηση και η κριτική τοποθέτηση των γενικών ποιοτικών χαρακτηριστικών τους.

Μετά τη μελέτη του κεφαλαίου αυτού, ο αναγνώστης αναμένεται ότι θα είναι σε θέση:

- να διακρίνει τα θεμελιώδη συστατικά της πειθαρχημένης ανάπτυξης λογισμικού τα οποία επιβάλλει η Τεχνολογία Λογισμικού,
- να περιγράφει τις έννοιες «μοντέλο κύκλου ζωής», «διαδικασία ανάπτυξης», «μεθοδολογία» και «εργαλείο», και να μπορεί να τις χρησιμοποιεί,
- να περιγράφει τα χαρακτηριστικά της ανάπτυξης λογισμικού όπως αυτά προτείνονται από τα επικρατέστερα μοντέλα κύκλου ζωής,
- να διακρίνει τα θετικά και τα αρνητικά στοιχεία της υιοθέτησης ενός μοντέλου κύκλου ζωής σε συγκεκριμένες συνθήκες,
- να διακρίνει ποιο μοντέλο κύκλου ζωής προσφέρεται για εφαρμογή σε μια δοσμένη περίπτωση κατασκευαστή λογισμικού και
- να διακρίνει περιγραφές των συστατικών στοιχείων του λογισμικού σύμφωνα με μια ταξινόμηση σε διαφορετικά επίπεδα λεπτομέρειας.

### Έννοιες-κλειδιά

---

- Μοντέλο κύκλου ζωής
- Δραστηριότητα ανάπτυξης λογισμικού
- Μεθοδολογία ανάπτυξης
- Εργαλείο λογισμικού
- Προδιαγραφή, ανάπτυξη, επαλήθευση, εξέλιξη λογισμικού

- Μοντέλο του καταρράκτη
- Μοντέλο πρωτοτυποποίησης
- Μοντέλο λειτουργικής επαύξησης
- Σπειροειδές μοντέλο
- Μοντέλο του πίδακα
- Καθολικό, εποπτικό, ατομικό επίπεδο περιγραφής

## Σύνοψη

---

Ενα μοντέλο κύκλου ζωής λογισμικού περιγράφει τις φάσεις από τις οποίες διέρχεται μια εφαρμογή λογισμικού από τη σύλληψη μέχρι την απόσυρσή της, καθώς και τις ενέργειες που λαμβάνουν χώρα σε καθεμία από αυτές. Μια δραστηριότητα ή διαδικασία ανάπτυξης λογισμικού (*software process*) καθορίζει ποιες ενέργειες πρέπει να γίνουν για να εκτελεστεί κάποια από τις φάσεις του κύκλου ζωής. Μια μεθοδολογία καθορίζει το πώς θα πρέπει να εκτελούνται οι διαδικασίες αυτές, δηλαδή ποιες επιμέρους ενέργειες περιλαμβάνουν, ποια βήματα γίνονται σε καθεμία, ποια προϊόντα παράγονται κ.ά. Η εκτέλεση των εργασιών μπορεί να υποστηρίζεται από ειδικά εργαλεία.

Τα μοντέλα κύκλου ζωής λογισμικού που έχουν παρουσιαστεί διακρίνονται σε ακολουθιακά και σε επαναληπτικά. Στα ακολουθιακά μοντέλα η ανάπτυξη γίνεται σε διαδοχικές διακριτές φάσεις και για ολόκληρο το σύστημα λογισμικού, ενώ στα επαναληπτικά η ανάπτυξη του λογισμικού γίνεται σε τμήματα. Χαρακτηριστικότερο ακολουθιακό μοντέλο είναι αυτό του καταρράκτη, ενώ το γενικότερο από τα επαναληπτικά είναι το σπειροειδές. Πρακτικά, χρησιμότερα στην πράξη είναι τα μοντέλα κύκλου ζωής που αφήνουν ελευθερία εξειδίκευσης στις εκάστοτε συνθήκες και δεν προσδιορίζουν με αυστηρότητα τις ενέργειες που πρέπει να γίνουν, τα προϊόντα κ.λπ. Δεν υπάρχει ένα «καλύτερο» μοντέλο κύκλου ζωής αλλά ένα καταλληλότερο στις εκάστοτε συνθήκες τόσο του κατασκευαστή, όσο και του θεματικού πεδίου της εφαρμογής λογισμικού.

Στον Πίνακα 2.1 που ακολουθεί φαίνονται συνοπτικά ορισμένα χαρακτηριστικά των μοντέλων κύκλου ζωής λογισμικού στα οποία θα αναφερθούμε στο κεφάλαιο αυτό.



Μοντέλο	Μέγεθος εφαρμογών	Μεταβολές στις απαιτήσεις	Προσαρμοστικότητα στον κατασκευαστή	Διάδοση
Καταρράκτη	Μικρό έως μεσαίο	Ανεπιθύμητες	Καμία	Μεγάλη με τάση μείωσης
Πρωτοτυποποίησης	Μικρό ως μεσαίο	Δεκτές	Μικρή	Μικρή με τάση αύξησης
Λειτουργικής επαύξησης	Μεσαίο ως μεγάλο	Ανεπιθύμητες	Καμία	Μικρή με τάση μείωσης
Σπειροειδές	Μεσαίο ως μεγάλο	Δεκτές	Αρκετή	Μικρή με τάση μείωσης
Πίδακα	Οποιοδήποτε	Δεκτές	Αρκετή	Μικρή
Γενικό	Οποιοδήποτε	Δεκτές	Μεγάλη	Μικρή με ισχυρές τάσεις αύξησης

### Εισαγωγικές παρατηρήσεις

Όπως είδαμε στο Κεφάλαιο I, η Τεχνολογία Λογισμικού είναι η περιοχή εκείνη της επιστήμης της πληροφορικής που ασχολείται με την εύρεση και θεμελίωση μεθόδων για να περιγράφεται, να κατασκευάζεται και να συντηρείται λογισμικό καλής ποιότητας με τη μεγαλύτερη δυνατή αυτοματοποίηση και παραγωγικότητα και το ελάχιστο δυνατό κόστος. Για να επιτύχει τους σκοπούς της, η Τεχνολογία Λογισμικού οφείλει να περιγράψει τις ενέργειες που πρέπει να συμβαίνουν κατά την ανάπτυξη του λογισμικού τόσο σε μακροσκοπικό, όσο και σε μικροσκοπικό επίπεδο. Σε μακροσκοπικό επίπεδο πρέπει να οριστούν οι γενικές φάσεις από τις οποίες διέρχεται η κατασκευή του λογισμικού, ενώ σε μικροσκοπικό πρέπει να οριστούν οι ενέργειες που γίνονται σε κάθε φάση και τα προϊόντα που παράγονται.

Δεν υπάρχει ένας και μοναδικός τρόπος για να προσδιορίσουμε έστω και τις γενικές φάσεις από τις οποίες διέρχεται η κατασκευή του λογισμικού. Όπως θα δούμε στο κεφάλαιο αυτό, πολλά εξαρτώνται από τις ιδιαίτερες συνθήκες που επικρατούν,

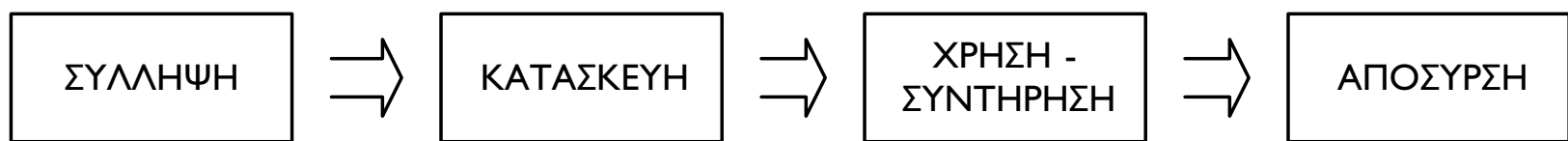


όπως η εμπειρία και η τεχνογνωσία του κατασκευαστή, το είδος της εφαρμογής λογισμικού και η πιθανότητα να αλλάξουν οι απαιτήσεις από αυτή κ.ά. Για τον λόγο αυτό έχουν προταθεί πολλές εναλλακτικές διαδρομές που μπορεί να ακολουθήσει κανείς στην κατασκευή και συντήρηση του λογισμικού, οι οποίες ονομάζονται «μοντέλα κύκλου ζωής». Στις χαρακτηριστικότερες από αυτές θα αναφερθούμε στο κεφάλαιο αυτό.

## **ΕΝΟΤΗΤΑ 2.1. Η ΕΝΝΟΙΑ ΤΟΥ ΜΟΝΤΕΛΟΥ ΚΥΚΛΟΥ ΖΩΗΣ**

Κάθε εφαρμογή λογισμικού, από τη σύλληψη μέχρι την απόσυρσή της, διέρχεται από διάφορες φάσεις σε καθεμία εκ των οποίων πρέπει να γίνονται ορισμένες εργασίες ώστε να επιτυγχάνεται το επιθυμητό αποτέλεσμα. Σε μακροσκοπικό επίπεδο οι πολύ γενικές φάσεις είναι: σύλληψη, κατασκευή, χρήση – συντήρηση και απόσυρση και, όπως είναι εύκολα αντιληπτό, λαμβάνουν χώρα με τη σειρά αυτή. Μια εικόνα των γενικών αυτών φάσεων φαίνεται στο Σχήμα 2.1 που ακολουθεί.

**Σχήμα 2.1** Γενικές φάσεις του κύκλου ζωής του λογισμικού.



Πριν αναφερθούμε στον ορισμό του μοντέλου κύκλου ζωής και στα σημαντικότερα τέτοια μοντέλα που χρησιμοποιούνται σήμερα, είναι σκόπιμο να δοθούν ορισμένοι χρήσιμοι ορισμοί οι οποίοι θα χρησιμοποιηθούν εκτεταμένα στη συνέχεια. Σε κάποιους από τους ορισμούς αυτούς ενδεχομένως ο αναγνώστης να συναντήσει και διαφορετικές απόψεις στη βιβλιογραφία. Όπως έχουμε ήδη αναφέρει, ο πλουραλισμός ελάχιστα διαφοροποιούμενων απόψεων δεν είναι σπάνιος στην Τεχνολογία Λογισμικού, η δε αναφορά πολλών διαφορετικών απόψεων σε αυτό το σημείο δεν εξυπηρετεί παρά τη σύγχυση.

### **Δραστηριότητα ανάπτυξης λογισμικού:**

Μια **δραστηριότητα** ή **διαδικασία ανάπτυξης λογισμικού** (software process) καθορίζει ποιες ενέργειες πρέπει να γίνουν για να επιτευχθεί ένα επιθυμητό αποτέλεσμα σε κάποια από τις φάσεις του κύκλου ζωής. Μια δραστηριότητα μπορεί να αναλύεται σε περισσότερες από μία **επιμέρους φάσεις**.

Η έννοια «ανάπτυξη» στον προηγούμενο ορισμό περιγράφει μια γενική διαδικασία στην οποία υπόκειται το λογισμικό και όχι υποχρεωτικά κατασκευή εκ του μηδενός.

### **Μεθοδολογία ανάπτυξης:**

Μια **μεθοδολογία** (software development methodology) καθορίζει το **πώς** θα πρέπει να εκτελούνται οι δραστηριότητες ανάπτυξης, δηλαδή ποιες επιμέρους ενέργειες περιλαμβάνουν, ποια βήματα γίνονται σε καθεμία, ποια προϊόντα παράγονται, καθώς και πότε αυτές θεωρούνται περατωθείσες.

### **Εργαλείο:**

Ένα **εργαλείο λογισμικού** είναι ένα σύστημα (συνήθως είναι και το ίδιο εφαρμογή λογισμικού) το οποίο υποστηρίζει τη μερική ή ολική αυτοματοποίηση των εργασιών που λαμβάνουν χώρα κατά την εφαρμογή των μεθοδολογιών ανάπτυξης λογισμικού.

Με βάση τα προηγούμενα, μπορούμε να δώσουμε τον ορισμό του μοντέλου κύκλου ζωής λογισμικού.

### **Μοντέλο κύκλου ζωής λογισμικού:**

Ένα μοντέλο κύκλου ζωής λογισμικού είναι μια περιγραφή των δραστηριοτήτων και των επιμέρους φάσεων από τις οποίες διέρχεται μια εφαρμογή λογισμικού από τη σύλληψη μέχρι την απόσυρσή της, καθώς και των εργασιών που λαμβάνουν χώρα σε καθεμία από τις φάσεις αυτές.

Στο Σχήμα 2.2 φαίνεται η σχέση μεταξύ των εννοιών «μοντέλο κύκλου ζωής», «διαδικασία ανάπτυξης», «μεθοδολογία», καθώς και «εργαλείο», οι οποίες ορίστηκαν προηγουμένως. Μια έννοια που βρίσκεται χαμηλότερα στην πυραμίδα αποτελεί το υπόβαθρο πάνω στο οποίο βασίζεται η έννοια που βρίσκεται στο αμέσως ψηλότερο σημείο κ.ο.κ.

**Σχήμα 2.2** Σχέσεις εννοιών στην ανάπτυξη του λογισμικού.



Τα μοντέλα κύκλου ζωής λογισμικού προσδιορίζουν τις διαδικασίες ανάπτυξης οι οποίες λαμβάνουν χώρα κατά τις γενικές φάσεις «κατασκευή» και «χρήση – συντήρηση» (Σχήμα 2.1), προσδιορίζοντας τις επιμέρους φάσεις στις οποίες αυτές αναλύονται, τα προϊόντα που παράγονται σε καθεμία από αυτές, καθώς και τη σειρά εκτέλεσής τους. Σε κάθε διαδικασία ανάπτυξης μπορούμε να διακρίνουμε περισσότερες από μία επιμέρους φάσεις, ενώ σε κάθε επιμέρους φάση μπορούμε να διακρίνουμε περισσότερες από μία εργασίες. Οι διαδικασίες ανάπτυξης λογισμικού μπορούν να ταξινομηθούν ως ακολούθως:

- **Προδιαγραφή**, δηλαδή καθορισμός των εργασιών που θα επιτελεί το λογισμικό, καθώς και των περιορισμών και των παραδοχών που ισχύουν.
- **Ανάπτυξη**, δηλαδή κατασκευή του λογισμικού. Εδώ, σε όλα τα μοντέλα κύκλου ζωής μπορούμε να διακρίνουμε τρεις επιμέρους φάσεις: την **ανάλυση**, τη **σχεδίαση** και τη **συγγραφή του πηγαίου κώδικα** (source code), την οποία στη συνέχεια θα ονομάζουμε και **κωδικοποίηση**.
- **Επαλήθευση**, δηλαδή επιβεβαίωση της ικανοποίησης των προδιαγραφών και της μη ύπαρξης σφαλμάτων.
- **Εξέλιξη**, δηλαδή επαύξηση των λειτουργικών χαρακτηριστικών του λογισμικού ή τροποποίηση υπάρχουσών, προκειμένου να ικανοποιούνται οι μεταβαλλόμενες ανάγκες.

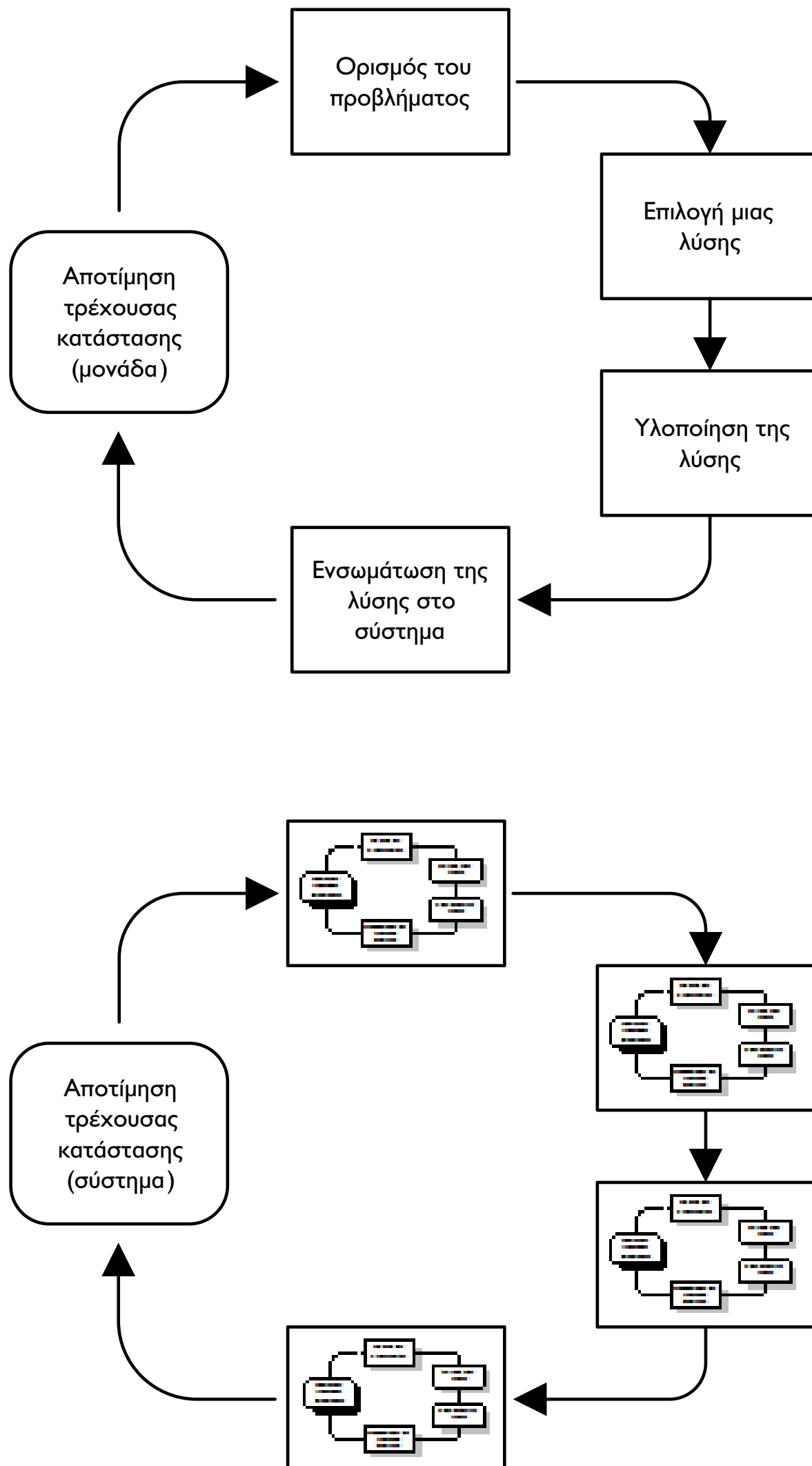
Ένα μοντέλο κύκλου ζωής λογισμικού στοχεύει στην καθοδήγηση του κατασκευαστή, προκειμένου αυτός να επιτύχει την καλύτερη δυνατή υλοποίηση των διαδικασιών ανάπτυξης λογισμικού. Όταν λέμε «καλύτερη δυνατή», εννοούμε περισσότερο παραγωγική, με τα λιγότερα δυνατά σφάλματα και το μικρότερο δυνατό ρίσκο στις εκάστοτε συνθήκες. Τα παραπάνω μπορούν να διαφοροποιούνται ανάλογα με το μέγεθος και το θεματικό πεδίο κάθε εφαρμογής λογισμικού, με την εμπειρία και τα ιδιαίτερα χαρακτηριστικά του κάθε κατασκευαστή και ασφαλώς με το εκάστοτε περιβάλλον ανάπτυξης.

Μια σημαντική παράμετρος που καταδεικνύει τη σημασία των μοντέλων κύκλου ζωής είναι το κόστος, ιδωμένο με την ευρύτερη σημασία του. Το κόστος αναθεώρησης αποφάσεων ή/και διόρθωσης σφαλμάτων είναι τόσο

μεγαλύτερο, όσο μεγαλύτερη είναι και η απαιτούμενη οπισθοδρόμηση της διαδικασίας που αυτή συνεπάγεται. Το κόστος αυτό δεν αφορά μόνο οικονομικούς πόρους που αποδίδονται στο έργο αλλά και χρόνο καθυστέρησης που δεν είναι πάντα διαθέσιμος σε πραγματικές συνθήκες. Επίσης, είναι συχνό φαινόμενο οι παρενέργειες στο υπόλοιπο σύστημα λογισμικού (side-effects) οι οποίες μπορούν να μεταβάλλουν προς το χειρότερο τα ποιοτικά του χαρακτηριστικά και δεν είναι εύκολο να εντοπιστούν από την αρχή.

Υπάρχουν αρκετά μοντέλα κύκλου ζωής τα οποία διαφοροποιούνται ως προς τη σύλληψη της ιδέας του τρόπου κατασκευής αλλά και ως προς τις επιμέρους φάσεις που προτείνουν, την επαναληπτικότητα και την εμβέλεια των εργασιών αυτών, τα ενδιάμεσα προϊόντα – συστατικά λογισμικού και την περιγραφή τους, τις οικονομικές και επιχειρηματικές πλευρές της χρήσης τους κ.ά. Καθεμία από τις ενέργειες που περιγράφεται σε ένα μοντέλο κύκλου ζωής είναι μια διαδικασία επίλυσης προβλημάτων (problem solving process). Τα βήματα κάθε τέτοιας διαδικασίας φαίνονται στο Σχήμα 2.3. Ο μηχανικός λογισμικού εκτελεί συνεχώς τέτοιες διαδικασίες επίλυσης προβλημάτων τόσο σε μικροσκοπικό επίπεδο, δηλαδή στις μονάδες του υπό κατασκευή λογισμικού (αριστερό τμήμα του σχήματος), όσο και σε μακροσκοπικό, δηλαδή για ολόκληρο το σύστημα (δεξί τμήμα του σχήματος).

**Σχήμα 2.3** Επίλυση προβλημάτων στο μικροσκοπικό (αριστερά) και μακροσκοπικό επίπεδο (δεξιά) ενός συστήματος λογισμικού.





Ακολούθως γίνεται μια σύντομη αναφορά στα σημαντικότερα μοντέλα κύκλου ζωής λογισμικού που έχουν εμφανιστεί με σκοπό την παρουσίαση της κεντρικής ιδέας και των ποιοτικών χαρακτηριστικών αυτών και όχι την αναλυτική παρουσίασή τους για την οποία ο αναγνώστης παραπέμπεται στη βιβλιογραφία. Στα μοντέλα κύκλου ζωής που θα παρουσιαστούν εμφανίζονται επιμέρους φάσεις και αντίστοιχες εργασίες με παρόμοια σειρά, μόνο που παριστάνονται με διαφορετικό σχηματικό τρόπο. Αυτό δεν σημαίνει ότι επαναλαμβάνονται τα ίδια πράγματα και ότι μόνη διαφορά είναι η σχηματική απεικόνιση. Ούτως ή άλλως, οι γενικές εργασίες κατά την ανάπτυξη του λογισμικού είναι οι τέσσερις που προαναφέρθηκαν, δηλαδή προδιαγραφή, ανάπτυξη, επαλήθευση, εξέλιξη.

Το πρώτο και γηραιότερο μοντέλο κύκλου ζωής λογισμικού, αυτό του καταρράκτη, αντιμετωπίζει την ανάπτυξη του λογισμικού σαν τη μεταφορά ενός μεγάλου ογκόλιθου από ένα σημείο σε κάποιο άλλο περνώντας από ενδιάμεσες στάσεις αλλά μεταφέροντας από τη μία στάση στην άλλη ολόκληρο τον ογκόλιθο. Επειδή οι εφαρμογές λογισμικού είναι περισσότερο εύπλαστες και ευμετάβλητες από τους ογκόλιθους, σύντομα παρουσιάστηκαν προβλήματα στην ιδέα, οπότε εμφανίστηκαν και άλλα μοντέλα κύκλου ζωής τα οποία μεταφέρουν με διαφορετικούς και πιο ευέλικτους τρόπους μικρότερα τμήματα του «ογκόλιθου».

Αυτό, λοιπόν, που διαφοροποιεί τα διάφορα μοντέλα κύκλου ζωής λογισμικού είναι η εμβέλεια, δηλαδή η έκταση του υπό κατασκευή συστήματος λογισμικού στην οποία αυτές οι διαδικασίες εφαρμόζονται, η επαναληπτικότητα των εργασιών, καθώς και οι ενδιάμεσες αποτιμήσεις από τον πελάτη ή τον κατασκευαστή. Έτσι, σε κάθε μοντέλο κύκλου ζωής είναι με διαφορετικό τρόπο δυνατός ο εντοπισμός της ανάγκης και η ενσωμάτωση τροποποιήσεων στα χαρακτηριστικά του λογισμικού προτού ολοκληρωθεί πλήρως η κατασκευή του, οπότε μπορεί να είναι αργά από πλευράς χρόνου και κόστους.

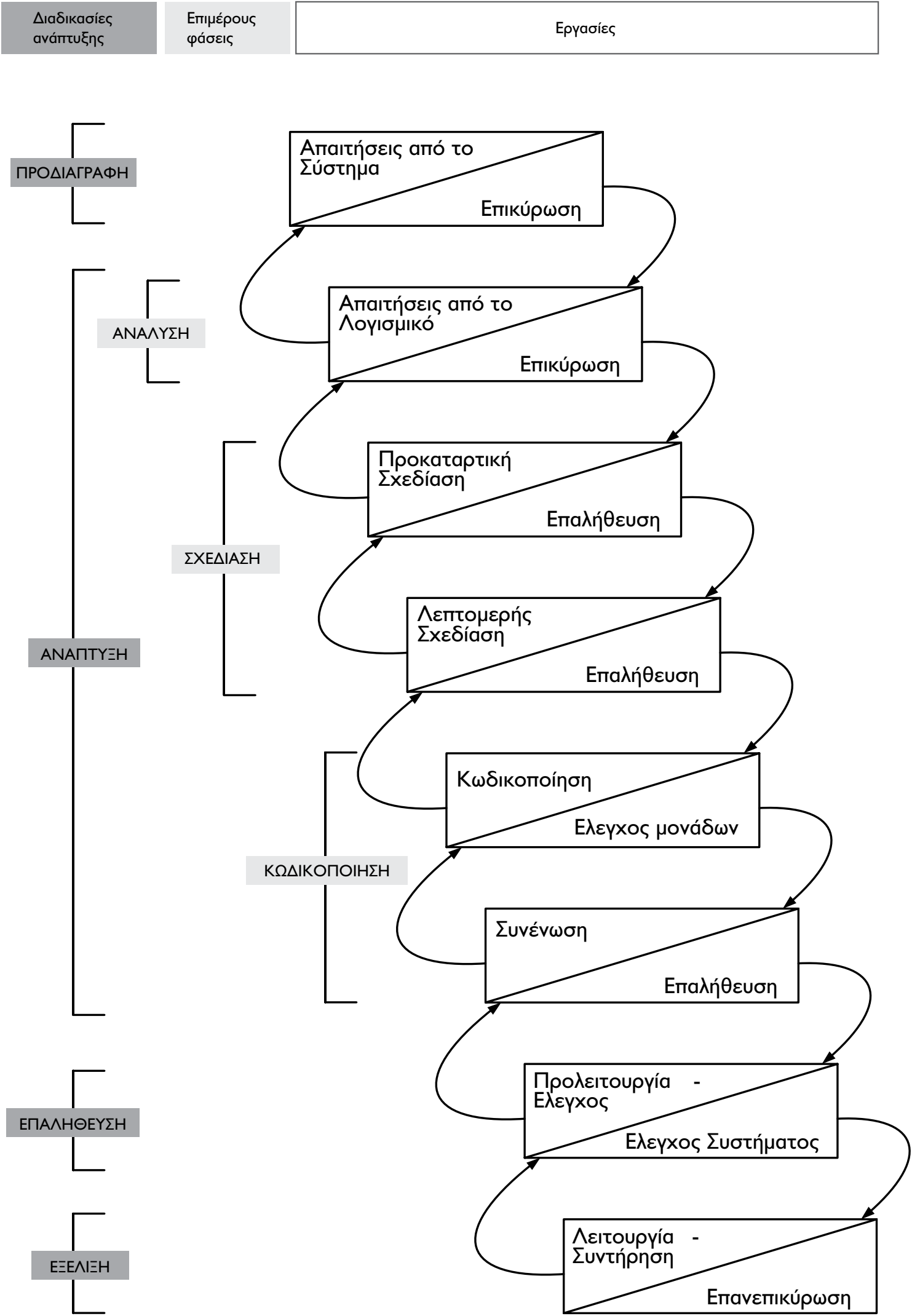
## Δραστηριότητα I/Κεφάλαιο 2

Προσπαθήστε να αναφέρετε τρεις διατυπώσεις που περιγράφουν την ποσοτική σχέση μεταξύ των εννοιών «μοντέλο κύκλου ζωής», «διαδικασία ανάπτυξης», «μεθοδολογία» και «εργαλείο». Θα χρειαστεί να προσπαθήσετε να συντάξετε προτάσεις του τύπου: «Ένα μοντέλο κύκλου ζωής μπορεί να υλοποιείται με περισσότερες από μία διαδικασίες ανάπτυξης». Μπορείτε να βοηθηθείτε από το Σχήμα 2.2 και να ανατρέξετε στις απαντήσεις για να δείτε και τη δική μας προσέγγιση.

## ΕΝΟΤΗΤΑ 2.2. ΤΟ ΜΟΝΤΕΛΟ ΤΟΥ ΚΑΤΑΡΡΑΚΤΗ

Ένα από τα πιο διαδεδομένα μοντέλα κύκλου ζωής είναι αυτό του καταρράκτη (Waterfall), το οποίο φαίνεται στο Σχήμα 2.4. Η κεντρική ιδέα του μοντέλου του καταρράκτη είναι ότι το σύστημα λογισμικού αναπτύσσεται περνώντας ολόκληρο από διαδοχικές επιμέρους φάσεις, καθεμία από τις οποίες θεωρείται περατωμένη με την παραγωγή ορισμένων συστατικών λογισμικού. Κάθε επιμέρους φάση ολοκληρώνεται με μια εργασία επαλήθευσης/επικύρωσης των προϊόντων της κατά την οποία αποφασίζεται η μετάβαση ή όχι στην επόμενη. Το λογισμικό εμφανίζεται πλήρες, δηλαδή με όλα τα λειτουργικά του χαρακτηριστικά, από την επιμέρους φάση της συνένωσης και μετά. Χαρακτηριστικό του μοντέλου του καταρράκτη είναι ότι για να ξεκινήσει μία φάση πρέπει να έχει ολοκληρωθεί πλήρως η προηγούμενη. Η ανάπτυξη με τον τρόπο αυτό χαρακτηρίζεται *ακολουθιακή*, διότι οι επιμέρους φάσεις από τις οποίες διέρχεται είναι διακριτές και ακολουθούν η μία την άλλη.

Σχήμα 2.4 Το μοντέλο του καταρράκτη.



Αρχικά, καθορίζονται οι **απαιτήσεις από το σύστημα** και το **λογισμικό** αντίστοιχα. Όπως αναφέρθηκε στο Κεφάλαιο I, το λογισμικό είναι μία μόνο από τις συνιστώσες του συστήματος, το οποίο μπορεί να περιλαμβάνει και άλλες ειδικές συσκευές, άλλες εφαρμογές λογισμικού κ.ά.

Έπειτα, γίνεται η **προκαταρτική** και η **λεπτομερής σχεδίαση** του λογισμικού αντίστοιχα. Κατά την **προκαταρτική σχεδίαση** καθορίζονται οι μονάδες που θα αποτελούν το λογισμικό, καθώς και οι συσχετίσεις μεταξύ τους. Ο καθορισμός αυτός μπορεί να γίνει σε περισσότερα από ένα επίπεδα λεπτομέρειας, ανάλογα με το μέγεθος και την πολυπλοκότητα του λογισμικού. Το πρώτο επίπεδο (αυτό με τη μικρότερη λεπτομέρεια) περιέχει τα υποσυστήματα, το δεύτερο περιέχει τις μονάδες μέσα σε κάθε υποσύστημα κ.ο.κ.

Κατά τη **λεπτομερή σχεδίαση** καθορίζεται η εσωτερική δομή κάθε μονάδας λογισμικού η οποία αντιστοιχεί πρακτικά σε μονάδες πηγαίου κώδικα προγράμματος. Ο καθορισμός αυτός περιλαμβάνει όλα τα απαραίτητα στοιχεία (αλγόριθμοι, δομές δεδομένων κ.λπ.), ώστε η **συγγραφή του πηγαίου κώδικα** που ακολουθεί να είναι μια διαδικασία διεκπεραίωσης και μόνο. Ακολουθεί η **συνένωση** των μονάδων σε σύστημα και ο έλεγχος του συστήματος, η ολοκλήρωση του οποίου επιτρέπει την παράδοση ολόκληρου του προϊόντος στον πελάτη και το πέρασμα στη φάση της **λειτουργίας και συντήρησης**.

Το μοντέλο του καταρράκτη υπήρξε για μεγάλο διάστημα το πιο διαδεδομένο μοντέλο κύκλου ζωής λογισμικού. Είναι ιδιαίτερα χρήσιμο σε περιπτώσεις όπου οι απαιτήσεις από το λογισμικό είναι από την αρχή γνωστές και δεν μεταβάλλονται κατά την ανάπτυξή του και μπορεί να χρησιμοποιηθεί αποτελεσματικά για τη βιομηχανοποίηση της ανάπτυξης τέτοιων εφαρμογών. Για παράδειγμα, τέτοιες είναι οι εφαρμογές που επιλύουν μεγάλα προβλήματα χρησιμοποιώντας μαθηματικούς υπολογισμούς. Σε πολλές, όμως, περιπτώσεις εφαρμογών οι απαιτήσεις είτε δεν είναι από την αρχή και με σαφήνεια γνωστές είτε ενδέχεται να μεταβληθούν κατά τη διάρκεια της ανάπτυξης.

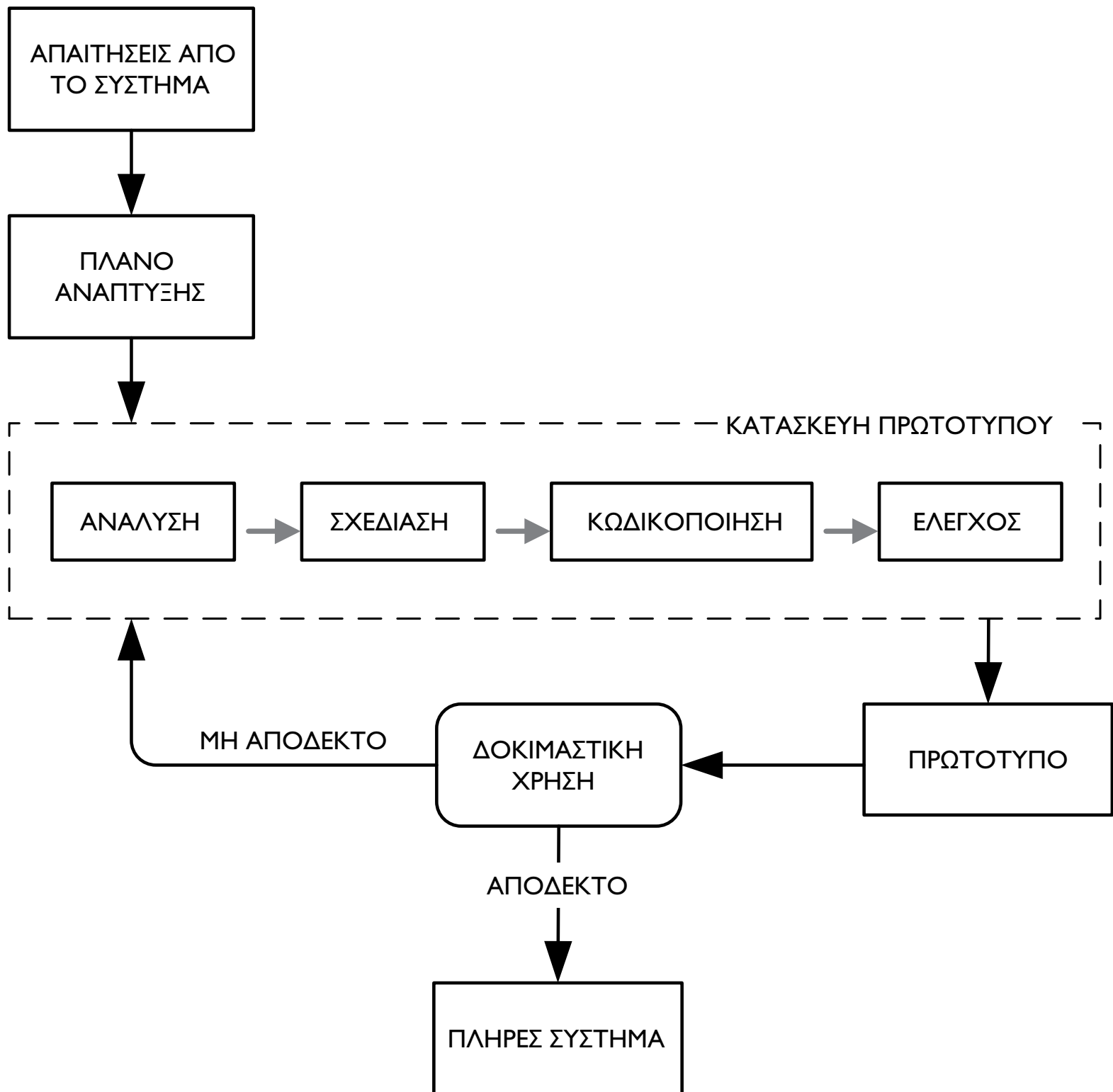
## Δραστηριότητα 2/Κεφάλαιο 2

Αναφέρατε τουλάχιστον δύο χαρακτηριστικά του μοντέλου του καταρράκτη τα οποία μπορούν να θεωρηθούν μειονεκτήματα με βάση τα όσα διαβάσατε στην προηγούμενη ενότητα. Όπως τονίσαμε και από την αρχή, το μοντέλο του καταρράκτη αντιμετωπίζει ολόκληρη την εφαρμογή λογισμικού σε κάθε βήμα. Σε αυτή την κατεύθυνση μπορείτε να εντοπίσετε αρκετά μειονεκτήματα και να τα συγκρίνετε με τη δική μας απάντηση στο τέλος του κεφαλαίου.

### ΕΝΟΤΗΤΑ 2.3. ΤΟ ΜΟΝΤΕΛΟ ΠΡΩΤΟΤΥΠΟΠΟΙΗΣΗΣ

Η κεντρική ιδέα του μοντέλου πρωτοτυποποίησης (prototyping model) είναι η ανάπτυξη του λογισμικού όχι εξ ολοκλήρου, αλλά σε τμήματα που ονομάζονται «πρωτότυπα». Οι διαδικασίες ανάπτυξης επαναλαμβάνονται για ένα τμήμα του συστήματος κάθε φορά και για το λόγο αυτό το μοντέλο χαρακτηρίζεται ως επαναληπτικό. Κάθε πρωτότυπο περιλαμβάνει τις βασικές από τις λειτουργίες που προορίζεται να εκτελεί το λογισμικό και τίθεται σε δοκιμασία από τον πελάτη. Από εκεί συλλέγονται παρατηρήσεις και η διαδικασία κατασκευής νέου πρωτοτύπου επαναλαμβάνεται μέχρις ότου ένα πρωτότυπο να ικανοποιεί τις απαιτήσεις, δηλαδή να εκτελεί τις επιθυμητές λειτουργίες του λογισμικού με τρόπο ικανοποιητικό και να γίνεται αποδεκτό από τον πελάτη (Σχήμα 2.5). Από το σημείο αυτό και μετά μπορούν να προστεθούν και οι υπόλοιπες λειτουργίες, ώστε το λογισμικό να ολοκληρωθεί.

**Σχήμα 2.5** Το μοντέλο της πρωτοτυποποίησης.



Ένα σημαντικό πλεονέκτημα του μοντέλου αυτού είναι η δυνατότητα απόκτησης άποψης για την εφαρμογή λογισμικού νωρίτερα απ' ό,τι στο μοντέλο του καταρράκτη. Αυτό μπορεί να γλιτώσει την ανάπτυξη από καθυστερήσεις (και συνεπαγόμενα κόστη) ή ακόμη και από ολική αποτυχία που θα επερχόταν αν ο κατασκευαστής αναγκαζόταν να οπισθοδρομήσει την ανάπτυξη ενώ αυτή είχε προχωρήσει πολύ. Παράλληλα, ιδιαίτερη σημασία αποκτά η διοίκηση του έργου η οποία πρέπει να εξασφαλίζει την υλοποιησιμότητα του πρωτοτύπου και την εύκολη τροποποίησή του. Κάθε κατασκευή πρωτοτύπου μπορεί να θεωρηθεί ως ένα μικρό έργο λογισμικού, το οποίο κατασκευάζεται με διαδικασίες που μπορούν να ακολουθούν άλλα μοντέλα κύκλου ζωής, όπως αυτό του καταρράκτη.

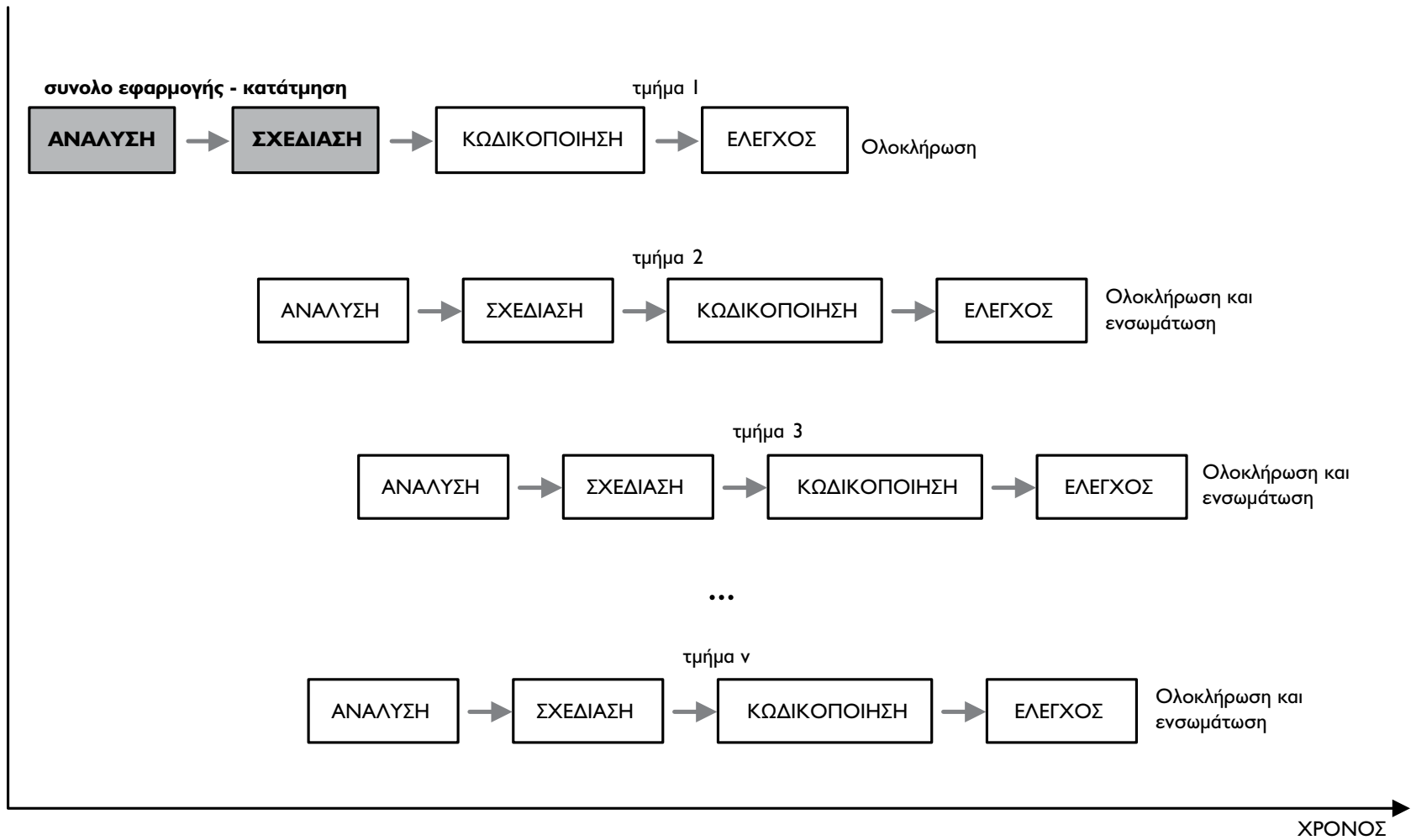
Με βάση τις παραπάνω παρατηρήσεις, το μοντέλο πρωτοτυποποίησης χρησιμοποιείται στην ανάπτυξη εφαρμογών λογισμικού για τις απαιτήσεις από τις οποίες δεν υπάρχει βεβαιότητα στην αρχή της ανάπτυξης, οπότε δεν μπορούν να συμφωνηθούν και να παγιωποιηθούν. Τέτοιες είναι εφαρμογές που κατασκευάζονται για πρώτη φορά ή που είναι στενά εξαρτημένες από τον πελάτη, χωρίς να υπάρχει αποδεκτό προηγούμενο παράδειγμα. Ωστόσο, το μέγεθος των εφαρμογών αυτών δεν μπορεί να είναι ιδιαίτερα μεγάλο, διότι ο χρόνος ανάπτυξης κάθε πρωτοτύπου μεγαλώνει και η απαιτούμενη ευελιξία μειώνεται.

## ΕΝΟΤΗΤΑ 2.4. ΤΟ ΜΟΝΤΕΛΟ ΛΕΙΤΟΥΡΓΙΚΗΣ ΕΠΑΥΞΗΣΗΣ

Το μοντέλο της λειτουργικής επαύξης (incremental model) συνδυάζει την ακολουθιακή ανάπτυξη του μοντέλου του καταρράκτη και την τμηματική ανάπτυξη του μοντέλου της πρωτοτυποποίησης. Κεντρική ιδέα είναι η κατάτμηση του υπό κατασκευή λογισμικού σε τμήματα που αναπτύσσονται ανεξάρτητα, ακολουθώντας το καθένα ακολουθιακή ανάπτυξη, σύμφωνα με το μοντέλο του καταρράκτη, όπως φαίνεται στο Σχήμα 2.6. Κατά την αρχική φάση ανάλυσης και σχεδίασης αποφασίζονται τα τμήματα στα οποία θα κατατμηθεί η εφαρμογή, η ανάπτυξη των οποίων γίνεται στη συνέχεια ανεξάρτητα και παράλληλα. Όταν ολοκληρώνεται η ανάπτυξη κάθε τμήματος, αυτό ενσωματώνεται στο σύνολο της εφαρμογής, διαδικασία η οποία δικαιολογεί και τον τίτλο «λειτουργική επαύξηση».



**Σχήμα 2.6** Το μοντέλο της λειτουργικής επαύξεσης.



Πλεονεκτήματα της ιδέας είναι η δυνατότητα παράλληλης ανάπτυξης, η οποία τελικά καταλαμβάνει μικρότερο χρόνο, καθώς και ο διαδοχικός εμπλουτισμός των λειτουργικών χαρακτηριστικών του λογισμικού. Τα βασικά μειονεκτήματα του μοντέλου είναι τα ακόλουθα:

- Η αρχική κατάτμηση και γενική σχεδίαση του συστήματος αποκτά ιδιαίτερη βαρύτητα. Σφάλματα σε αυτή μπορούν να έχουν σημαντικές επιπτώσεις στο λογισμικό που θα κατασκευαστεί στη συνέχεια.
- Σε περίπτωση μεταβολής των λειτουργικών απαιτήσεων κατά τη χρήση του ημιτελούς συστήματος μπορεί η αρχιτεκτονική αυτού να μεταβληθεί σε βαθμό που να κλονιστεί η ανάπτυξη των υπόλοιπων τμημάτων αυτού.

Το μοντέλο της λειτουργικής επαύξησης χρησιμοποιείται στην ανάπτυξη μεγάλων εφαρμογών λογισμικού για τις οποίες ισχύουν οι απαιτήσεις του μοντέλου του καταρράκτη, δηλαδή σαφής γνώση και μικρή ή καθόλου μεταβλητότητα των απαιτήσεων κατά την ανάπτυξη.

### **Άσκηση I/Κεφάλαιο 2**

**Ποιο από τα χαρακτηριστικά της ανάπτυξης λογισμικού που αναφέρθηκαν στο Κεφάλαιο I του βιβλίου αυτού πιστεύετε ότι μπορεί να κλονίσει την επάρκεια του μοντέλου της λειτουργικής επαύξησης και να συμβάλει στην ανάδειξη των μειονεκτημάτων του;**

## ΕΝΟΤΗΤΑ 2.5. ΤΟ ΣΠΕΙΡΟΕΙΔΕΣ ΜΟΝΤΕΛΟ

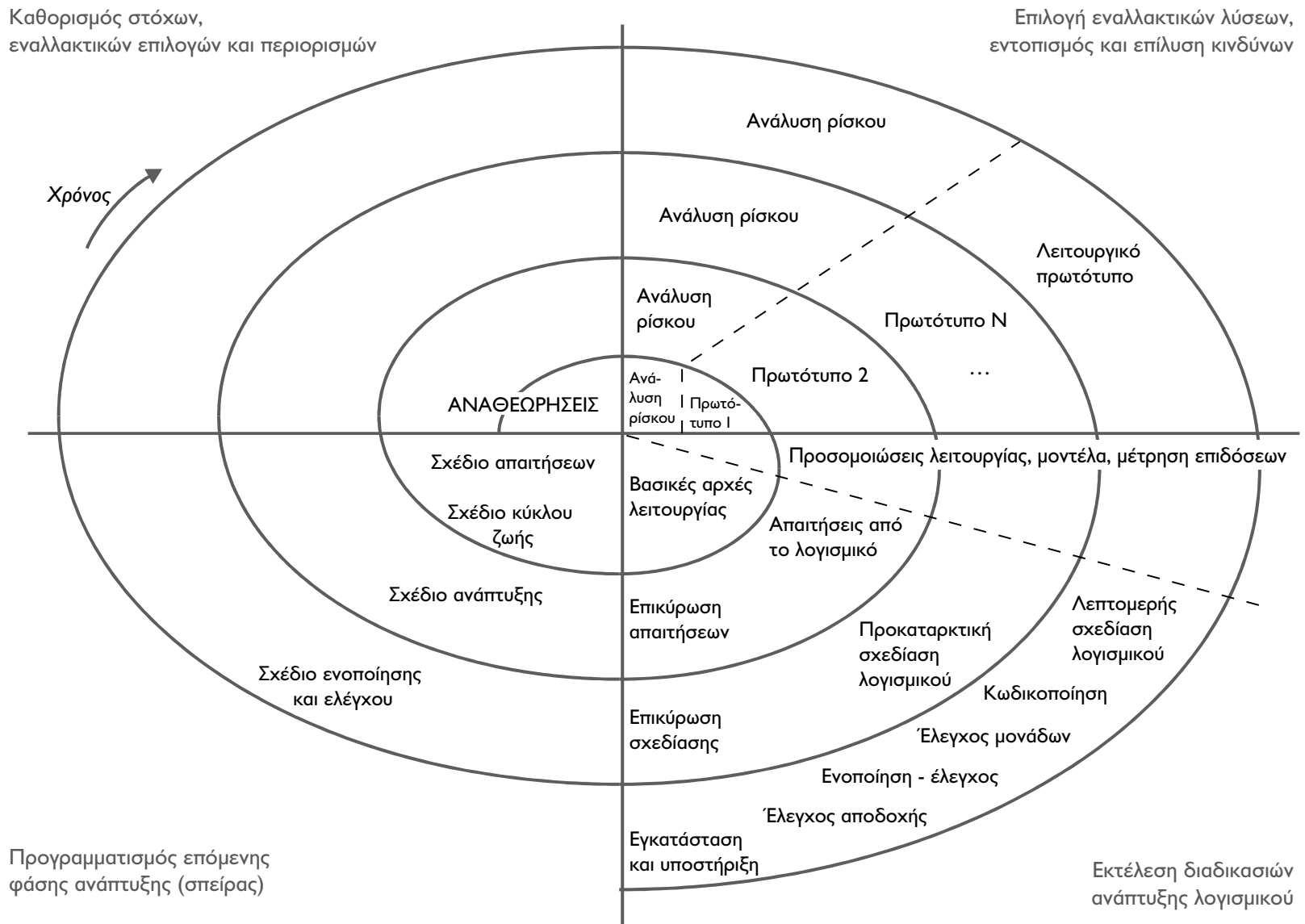
Τα μοντέλα κύκλου ζωής που παρουσιάστηκαν μέχρι τώρα αποτελούν παραλλαγές της βασικής ιδέας του μοντέλου του καταρράκτη. Η ανάπτυξη παραμένει επί της ουσίας μια ακολουθιακή διαδικασία η οποία εφαρμόζεται είτε σε ολόκληρο είτε σε ένα μέρος του συστήματος. Απ' ό,τι φαίνεται, δεν είναι η σύλληψη των διαδικασιών ανάπτυξης λογισμικού που διαφοροποιεί τα μοντέλα κύκλου ζωής, αλλά η διάταξή τους. Στο μοντέλο της πρωτοτυποποίησης, καθώς και σε αυτό της λειτουργικής επαύξησης, η κατάτμηση είναι λίγο έως πολύ αυθαίρετη. Το ρίσκο δεν αποτιμάται, με αποτέλεσμα κάθε οπισθοδρόμηση ή ανατροπή να κοστίζει σε χρόνο και σε οικονομικούς όρους, συχνά δε σε συνολική αποτυχία των έργων.

Από την άλλη, η μετά πειθαρχίας αποδοχή των αυστηρών φάσεων που προτείνονται από το μοντέλο του καταρράκτη δεν είναι εφικτό να ακολουθείται σε όλες τις περιπτώσεις και από όλους τους κατασκευαστές, με αποτέλεσμα η ανάπτυξη λογισμικού είτε να γίνεται άναρχα με βάση τη διαίσθηση των κατασκευαστών είτε να είναι μια δαπανηρή και στριφνή διαδικασία στην οποία «πρέπει» να ακολουθηθούν κάποια συγκεκριμένα βήματα ανεξάρτητα από τις εκάστοτε συνθήκες.

Απάντηση στα παραπάνω δίνει το σπειροειδές μοντέλο κύκλου ζωής, το οποίο πήρε το όνομά του από την απεικόνιση σε διάγραμμα, όπως φαίνεται στο Σχήμα 2.7. Πρόκειται για μια γενίκευση των μοντέλων της λειτουργικής επαύξησης και της πρωτοτυποποίησης με σημαντικά νέα στοιχεία:

- Οι φάσεις και οι διαδικασίες ανάπτυξης λογισμικού δεν είναι προκαθορισμένες από το μοντέλο, αλλά εξειδικεύονται στο χώρο της εφαρμογής του.
- Η ανάπτυξη ολόκληρου του συστήματος χωρίζεται σε πολλούς κύκλους, σε καθέναν από τους οποίους προστίθενται νέα λειτουργικά χαρακτηριστικά στο σύστημα.
- Πριν από την έναρξη κάθε κύκλου γίνεται μια μελέτη σκοπιμότητας και ανάλυση κινδύνων από την οποία προκύπτουν, αφενός, οι συγκεκριμένες εργασίες που θα εκτελεστούν μέσα στον κύκλο και, αφετέρου, η ίδια η εφικτότητα εκτέλεσης του κύκλου αυτού.

**Σχήμα 2.7** Το σπειροειδές μοντέλο κύκλου ζωής λογισμικού.



Όπως φαίνεται στο Σχήμα 2.7, στο σπειροειδές μοντέλο διακρίνονται τέσσερις κατηγορίες εργασιών: **προσδιορισμός στόχων, εντοπισμός και επίλυση κινδύνων, εκτέλεση διαδικασιών ανάπτυξης και επαλήθευση**, καθώς και **εργασίες προγραμματισμού**.

- Κατά τον **προσδιορισμό στόχων** καθορίζονται τα αντικείμενα εργασιών κάθε επανάληψης και καταγράφονται οι περιορισμοί επί του προϊόντος αλλά και επί της διαδικασίας, για την οποία κατασκευάζεται ένα αναλυτικό πλάνο διοίκησης. Επίσης, καταγράφονται οι κίνδυνοι που εμπεριέχει η διαδικασία και οι εναλλακτικές λύσεις, όπου υπάρχουν.
- Κατά τις εργασίες **επίλυσης κινδύνων** αναλύονται οι κίνδυνοι που έχουν καταγραφεί και αποτιμάται κάθε εναλλακτική λύση. Στο σημείο αυτό λαμβάνονται αποφάσεις για τη συνέχιση ή όχι της ανάπτυξης, για το μοντέλο που θα ακολουθηθεί στη συγκεκριμένη επανάληψη, για την κατασκευή ή όχι πρωτοτύπου κ.ά.
- Ακολουθεί η **εκτέλεση** των βημάτων της **διαδικασίας ανάπτυξης** λογισμικού που έχει επιλεγεί για το τμήμα εκείνο του συστήματος που αφορά η τρέχουσα επανάληψη.
- Τέλος, μετά την επαλήθευση των αποτελεσμάτων – ενδιάμεσων προϊόντων λογισμικού, γίνεται **προγραμματισμός** της συνέχισης της ανάπτυξης.

Το σπειροειδές μοντέλο δεν καθορίζει εκ των προτέρων ποιες ακριβώς είναι οι εργασίες ανάπτυξης λογισμικού που πρέπει να γίνουν ούτε σε ποια έκταση του συστήματος αυτές θα εφαρμοστούν. Διαφορετικές διαδικασίες ανάπτυξης μπορεί να επιλεγούν για διαφορετικά τμήματα του λογισμικού. Αυτό που προτείνει είναι ότι ο καθορισμός των λεπτομερειών υλοποίησης πρέπει να γίνεται συνεχώς κατά την ανάπτυξη (και όχι μία φορά, όπως συμβαίνει με τα μοντέλα κύκλου ζωής που αναφέρθηκαν μέχρι τώρα) με ευθύνη και με τεκμηρίωση από πλευράς του ίδιου του κατασκευαστή.

Η εφαρμογή του σπειροειδούς μοντέλου στην πράξη δεν είναι πάντα εύκολη υπόθεση. Εισάγονται νέες εργασίες που δεν ανήκουν καθαρά στις εργασίες ανάπτυξης λογισμικού αλλά αφορούν την τεκμηρίωση της σκοπιμότητας και τον τμηματικό προγραμματισμό της ανάπτυξης. Οι εργασίες αυτές

επιφέρουν ασφαλώς κάποιο κόστος, το οποίο όμως μπορεί να αποσβεστεί από τον έγκαιρο εντοπισμό προβλημάτων και την αποφυγή πιθανού ναυαγίου, κάτι που έχει συμβεί σε αρκετές περιπτώσεις.

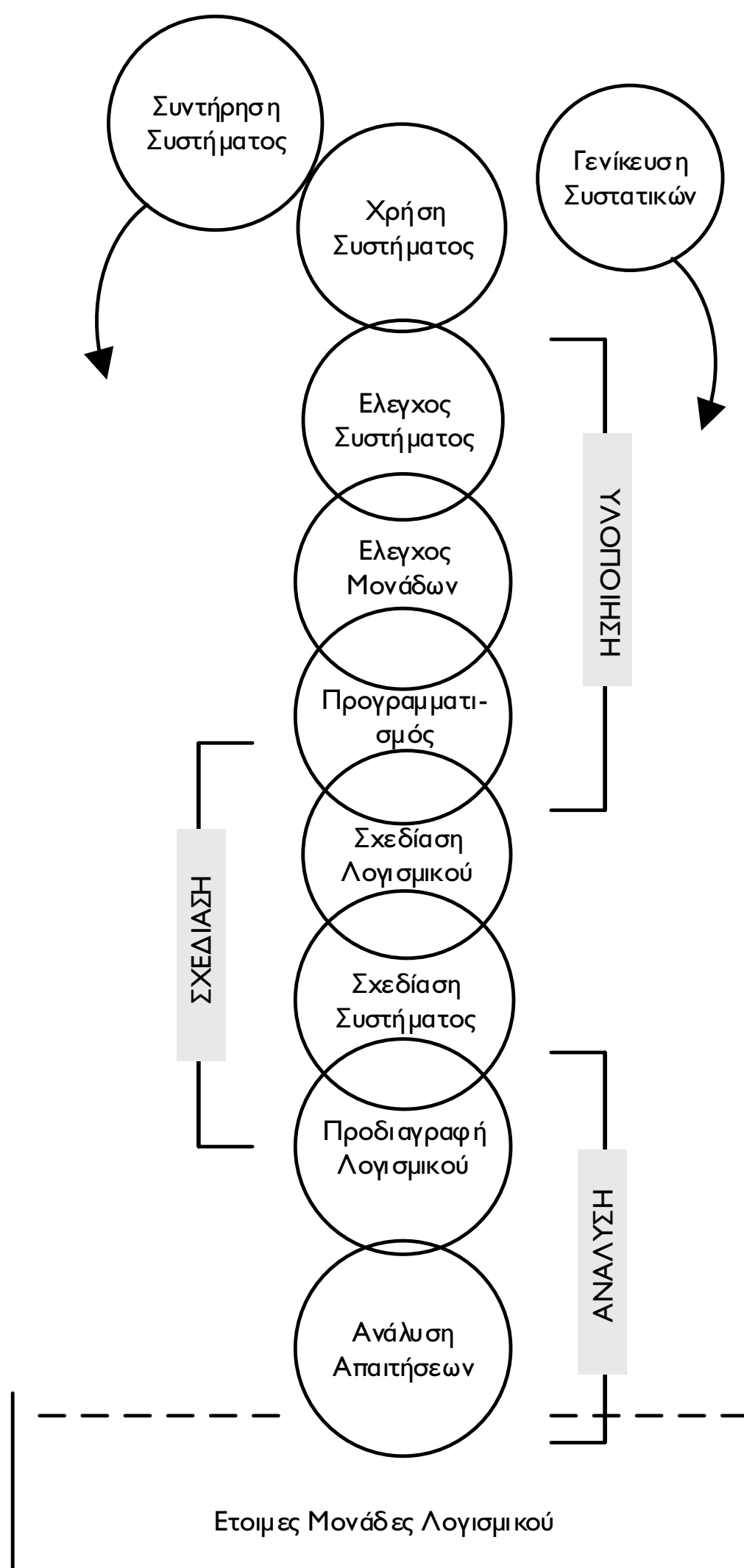
## Άσκηση 2/Κεφάλαιο 2

**Σχολιάστε με 80-120 λέξεις την καταλληλότητα του σπειροειδούς μοντέλου στην υλοποίηση μικρών έργων ανάπτυξης λογισμικού. Μπορείτε να στραφείτε στην κατεύθυνση της αναποτελεσματικότητας της άσκησης διοίκησης, όταν η ίδια είναι δυσανάλογα δαπανηρή σε σχέση με το αντικείμενο αυτής.**

## ΕΝΟΤΗΤΑ 2.6. ΤΟ ΜΟΝΤΕΛΟ ΤΟΥ ΠΙΔΑΚΑ

Αρκετά μοντέλα κύκλου ζωής που έχουν προταθεί αποτελούν παραλλαγές αυτών που αναφέρθηκαν, τα χαρακτηριστικά των οποίων υποβάλλονται από τις μεθοδολογίες ανάπτυξης. Οι πρώτες προσεγγίσεις του θέματος με βάση την αντικειμενοστρεφή (object-oriented) τεχνολογία διαφοροποίησαν το παραπάνω σχήμα βασιζόμενες σε δύο ιδιαίτερα γνωρίσματά της: πρώτον, ότι οι έννοιες «ανάλυση», «σχεδίαση», «κωδικοποίηση» έρχονται στο αντικειμενοστρεφές παράδειγμα πολύ πιο κοντά και, δεύτερον, ότι το αποτέλεσμα κάθε διαδικασίας κατασκευής λογισμικού είναι όχι μόνο ένα σύστημα αλλά και επαναχρησιμοποιήσιμες μονάδες οι οποίες μπορούν να χρησιμοποιηθούν από τις πρώτες φάσεις της ανάπτυξης μελλοντικών συστημάτων. Με τον τρόπο αυτό προέκυψε το μοντέλο του πίδακα (fountain model) που φαίνεται στο Σχήμα 2.8.

**Σχήμα 2.8** Το μοντέλο κύκλου ζωής του πίδακα, το οποίο βασίζεται στην αντικειμενοστρεφή τεχνολογία ανάπτυξης λογισμικού.



Κατά την ανάπτυξη παρατηρούνται επικαλύψεις των φάσεων «ανάλυση», «σχεδίαση», «κωδικοποίηση», οι οποίες φαίνονται με την επικάλυψη των κύκλων στο σχήμα. Κατά το τέλος της ανάπτυξης ορισμένα από τα συστατικά του λογισμικού που έχουν παραχθεί ενσωματώνονται σε μια «δεξαμενή» συστατικών και διατίθενται για να χρησιμοποιηθούν στην ανάπτυξη και νέων συστημάτων. Η ιδέα του μοντέλου κύκλου ζωής του πίδακα τονίζει περισσότερο τα επιθυμητά χαρακτηριστικά της μεθοδολογίας κατασκευής του λογισμικού σύμφωνα με την αντικειμενοστρεφή λογική, ήταν δε αρκετά επίκαιρη κατά την έκρηξη ενδιαφέροντος για την αντικειμενοστρεφή τεχνολογία στα τέλη της δεκαετίας του '80 και στις αρχές της δεκαετίας του '90.

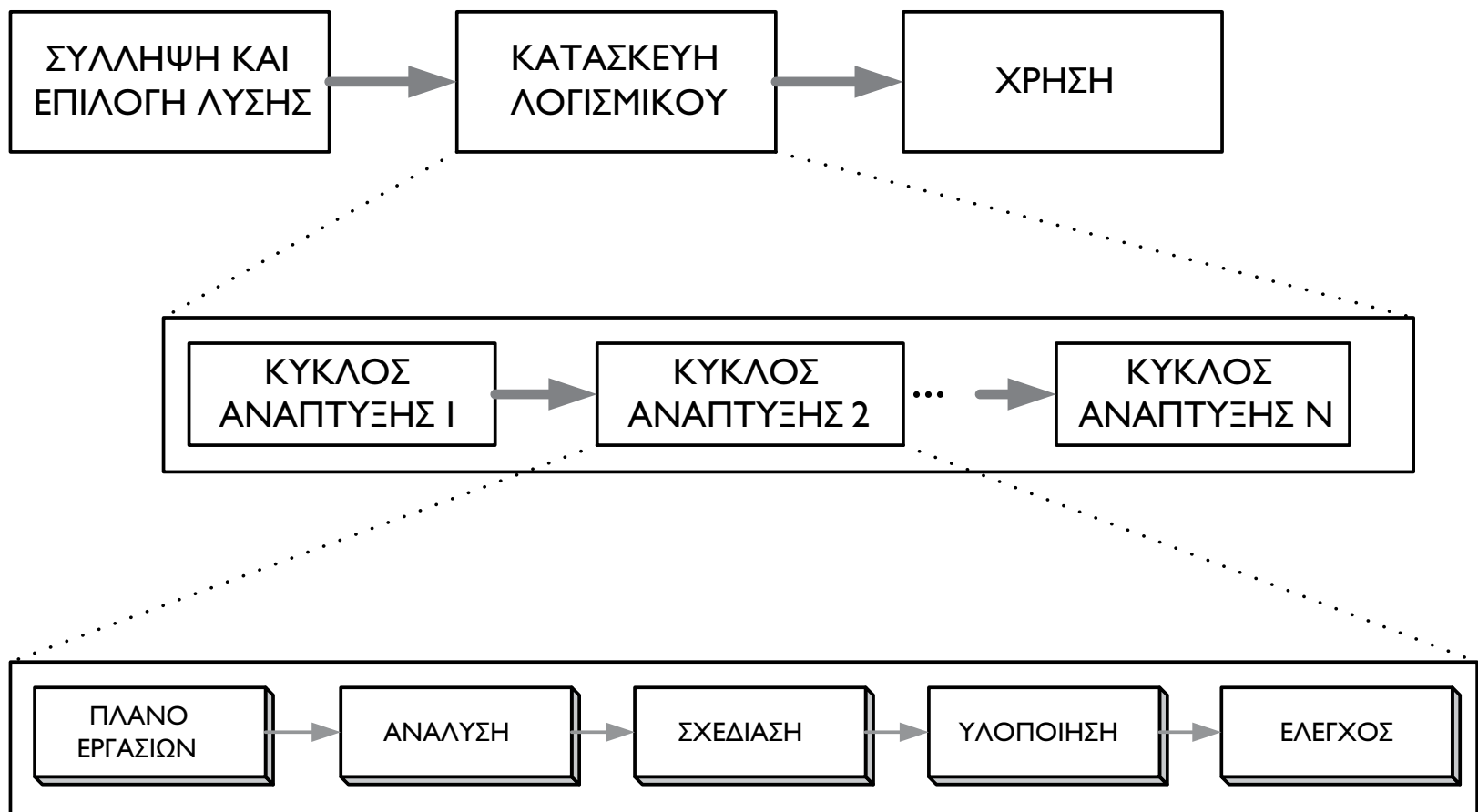


## ΕΝΟΤΗΤΑ 2.7. ΣΥΓΧΡΟΝΑ ΜΟΝΤΕΛΑ ΚΥΚΛΟΥ ΖΩΗΣ ΛΟΓΙΣΜΙΚΟΥ

Μεταγενέστερα μοντέλα κύκλου ζωής λογισμικού προσπαθούν να δώσουν μια γενική κατεύθυνση εφαρμογής των υπάρχουσών ιδεών, αφήνοντας σημαντικούς βαθμούς ελευθερίας στον κατασκευαστή που τα ακολουθεί. Αυτό είναι ιδιαίτερα επιθυμητό, διότι η αυστηρή πειθαρχία που επιχειρήθηκε να εισαχθεί τα πρώτα χρόνια της έκρηξης της χρήσης του λογισμικού δεν συμβάδιζε με την ωριμότητα σκέψης που διέθετε η τεχνική κοινότητα την εποχή εκείνη ούτε και μπορούσε να παρακολουθήσει τους υψηλούς ρυθμούς εξελίξεων στο χώρο της πληροφορικής. Χαρακτηριστικό είναι ότι συχνά ένα πολύ μεγάλο μέρος ογκοδέσφατων παραδοτέων (σχεδίων, προδιαγραφών κ.λπ.) δεν ήταν παρά λευκές σελίδες με τη μόνη ένδειξη «this page has been intentionally left blank», οι οποίες όμως ήταν υποχρεωτικό να υπάρχουν σύμφωνα με το ακολουθούμενο μοντέλο ανάπτυξης. Η πειθαρχία αυτή τελικά δεν οδήγησε στην κατασκευή λογισμικού αναμενόμενης ποιότητας.

Μια περιγραφή ενός σύγχρονου μοντέλου κύκλου ζωής λογισμικού περιέχει μόνο γενικές κατευθύνσεις, οι οποίες εξειδικεύονται στο εκάστοτε περιβάλλον ανάπτυξης, πρόβλημα κ.λπ. Επίσης, δεν είναι άρρηκτα συνδεδεμένο με κάποια μεθοδολογία ανάπτυξης λογισμικού αλλά μπορεί να εξειδικευτεί για την πρακτική του κάθε κατασκευαστή. Ένα τέτοιο μοντέλο φαίνεται στο Σχήμα 2.9 και μπορεί να χαρακτηριστεί ως απόγονος πολλών μοντέλων που προαναφέρθηκαν.

**Σχήμα 2.9** Ένα γενικό μοντέλο κύκλου ζωής το οποίο ενσωματώνει χαρακτηριστικά πολλών μοντέλων που αναφέρθηκαν.



Το γενικό πλαίσιο του μοντέλου αυτού περιλαμβάνει τις φάσεις σύλληψης, κατασκευής και λειτουργίας. Καθεμία από αυτές αναλύεται σε επιμέρους εργασίες, σύμφωνα με τα χαρακτηριστικά του εκάστοτε περιβάλλοντος. Ιδιαίτερα, η γενική φάση της κατασκευής αναλύεται σε κύκλους ανάπτυξης, καθένας εκ των οποίων προσθέτει νέα χαρακτηριστικά και λειτουργίες στο υπό κατασκευή λογισμικό.

Τα επιμέρους βήματα μέσα σε κάθε κύκλο ανάπτυξης μοιάζουν με τα βήματα του μοντέλου του καταρράκτη μόνο που δεν εφαρμόζονται για ολόκληρο το σύστημα αλλά για το μικρό μέρος του που κατασκευάζεται στον εν λόγω κύκλο, όπως στο μοντέλο της πρωτοτυποποίησης. Για την εκκίνηση κάθε κύκλου ανάπτυξης μπορεί να έχει προηγηθεί ανάλυση ρίσκου και σκοπιμότητας, όπως στο σπειροειδές μοντέλο. Ζητήματα όπως αλληλουχία των ενεργειών, ακριβής καθορισμός των κύκλων ανάπτυξης κ.ά. αφήνονται στη διακριτική ευχέρεια του κάθε κατασκευαστή από τον οποίο και καθορίζονται, σύμφωνα με τις ιδιαιτερότητες κάθε περίπτωσης. Ένα πραγματικό τέτοιο πλαίσιο ανάπτυξης προτείνεται από τη μεθοδολογία Objectory, η οποία είναι το προϊόν σύγκλισης των επικρατέστερων αντικειμενοστρεφών μεθοδολογιών ανάπτυξης λογισμικού.

### **Δραστηριότητα 3/Κεφάλαιο 2**

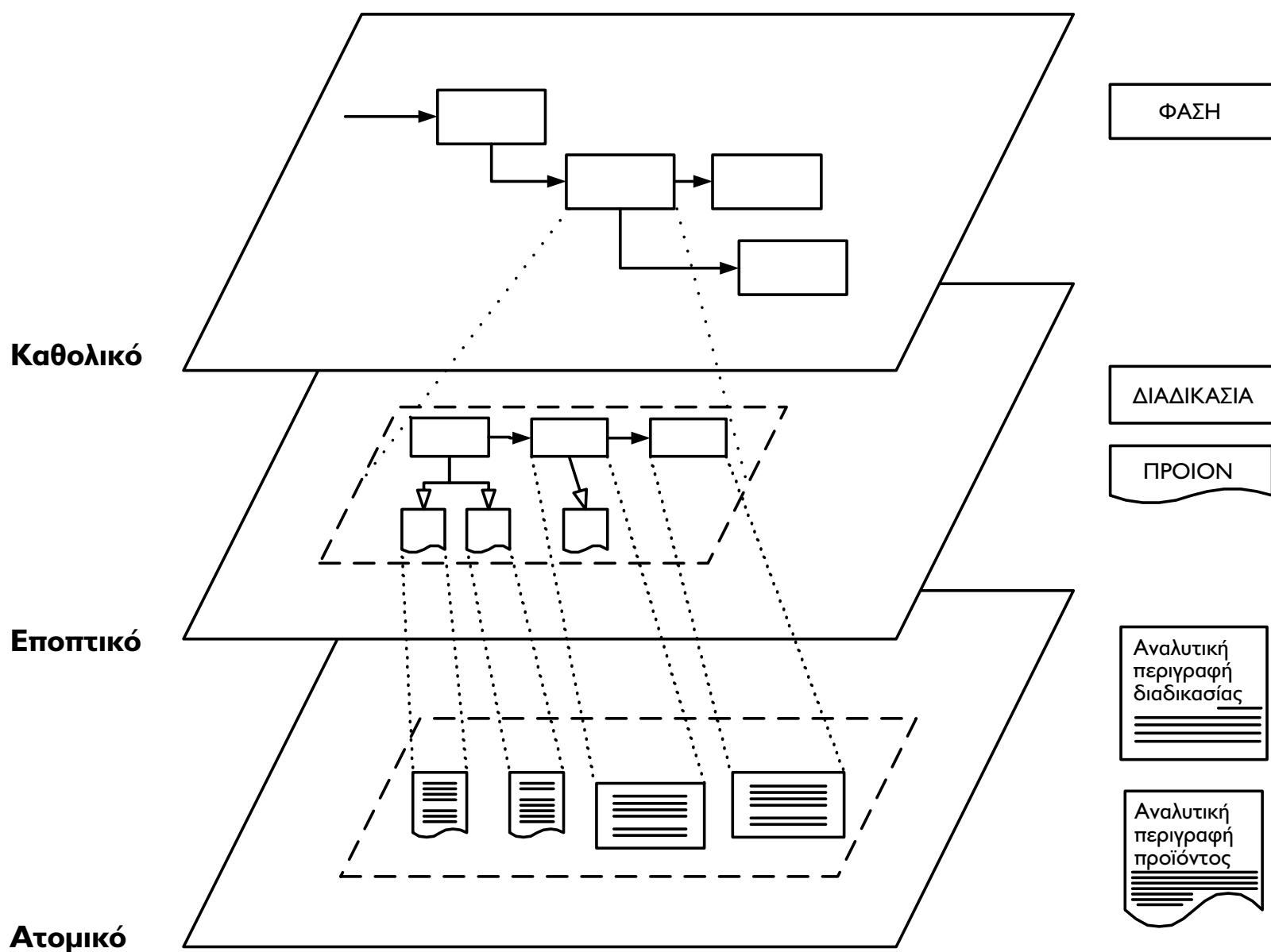
Επαληθεύσατε με σύντομους ποιοτικούς συλλογισμούς τη γενικότητα του μοντέλου που παρουσιάστηκε στην προηγούμενη ενότητα, δηλαδή τη δυνατότητα όλα τα μοντέλα κύκλου ζωής που αναφέρθηκαν να μπορούν να θεωρηθούν ειδικές εκδοχές αυτού.

## ΕΝΟΤΗΤΑ 2.8. ΣΥΓΧΡΟΝΑ ΜΟΝΤΕΛΑ ΚΥΚΛΟΥ ΖΩΗΣ ΛΟΓΙΣΜΙΚΟΥ

Η περιγραφή των διαδικασιών ανάπτυξης λογισμικού μπορεί να γίνει με μεγαλύτερη ή μικρότερη λεπτομέρεια, όπως φαίνεται στο Σχήμα 2.10. Η επιλογή της λεπτομέρειας με την οποία θα γίνει η περιγραφή αυτή εξαρτάται από το σκοπό της περιγραφής, τους αποδέκτες αυτής και γενικότερα από τις εκάστοτε συνθήκες. Είναι προφανές ότι με άλλη λεπτομέρεια θέλει (και μπορεί) να βλέπει την ανάπτυξη του λογισμικού ένας διοικητής έργου και με άλλη ένα μέλος μιας ομάδας ανάπτυξης.

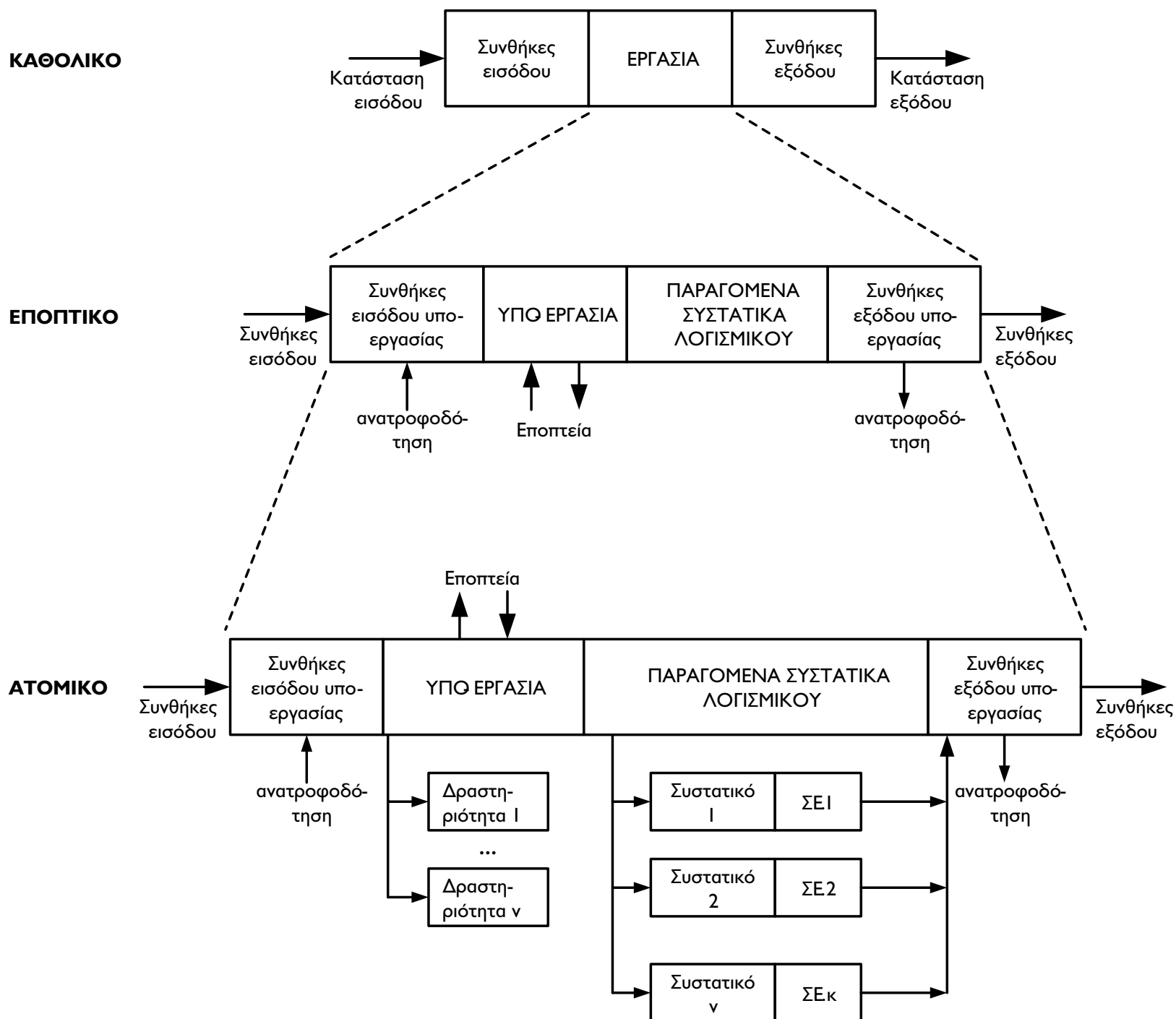
Τα επίπεδα λεπτομέρειας που μπορούμε να διακρίνουμε στην περιγραφή αυτή είναι τρία: το **καθολικό**, όπου περιέχεται μια γενική εποπτεία της διαδικασίας, το **εποπτικό**, όπου για κάθε γενικό βήμα περιέχονται σε γενική αναφορά οι επιμέρους εργασίες και τα προϊόντα και, τέλος, το **ατομικό**, όπου όλες οι διαδικασίες και τα προϊόντα αποκτούν πλήρεις αναλυτικές περιγραφές. Η παρουσίαση των μοντέλων κύκλου ζωής στο κεφάλαιο αυτό έγινε στο καθολικό επίπεδο.

**Σχήμα 2.10** Επίπεδα λεπτομέρειας στην περιγραφή διαδικασιών ανάπτυξης λογισμικού κατά H. Watt.



Σε κάθε επίπεδο η περιγραφή γίνεται με διαφορετική λεπτομέρεια, όπως φαίνεται στο Σχήμα 2.ΙΙ. Η περιγραφή στο ατομικό επίπεδο περιέχει τη μεγαλύτερη λεπτομέρεια, ενώ η περιγραφή στο καθολικό περιέχει τη μικρότερη. Η περιγραφή της κατάστασης τη στιγμή της εισόδου σε μια φάση του κύκλου ζωής (κατάσταση εισόδου) ελέγχεται ως προς την ικανοποίηση ορισμένων συνθηκών εισόδου (συνθήκες εισόδου). Μετά την ολοκλήρωση των εργασιών που περιληπτικά περιγράφονται στο καθολικό επίπεδο, ικανοποιείται ένα σύνολο συνθηκών εξόδου οι οποίες καθορίζουν την κατάσταση εξόδου από τη φάση. Ανάλογα με τη φύση των εργασιών που γίνονται στη φάση του κύκλου ζωής που περιγράφει το σχήμα αυτό, υπάρχει η δυνατότητα λήψης και παραγωγής πληροφοριών ανατροφοδότησης (feedback) προς άλλες φάσεις του κύκλου ζωής.

**Σχήμα 2.11** Πληροφορίες που περιέχονται σε κάθε επίπεδο λεπτομέρειας κατά την περιγραφή διαδικασιών ανάπτυξης λογισμικού.



Στο εποπτικό επίπεδο καθεμία από τις φάσεις του καθολικού επιπέδου αναλύεται σε επιμέρους εργασίες, σε καθεμία από τις οποίες παράγονται κάποια προϊόντα. Από το σύνολο των συνθηκών που περιγράφουν την είσοδο στη φάση του κύκλου ζωής, καθώς και την έξοδο από αυτή, μόνο ένα υποσύνολο (στη γενική περίπτωση) αφορά κάθε υποεργασία. Το ίδιο ισχύει και για τις πληροφορίες υποστήριξης, ενώ, επιπλέον, στο σημείο αυτό αναγνωρίζονται και πληροφορίες που αφορούν την ίδια την εκτέλεση της υποεργασίας (tracking). Τέλος, στο ατομικό επίπεδο περιλαμβάνονται επιπλέον αναλυτικές περιγραφές των δραστηριοτήτων και των προϊόντων που παράγονται σε κάθε επιμέρους εργασία.

Κάθε παραγόμενο προϊόν δύναται να επιδρά στην ικανοποίηση των συνθηκών εξόδου και θα πρέπει να περιγράφεται με επαρκή λεπτομέρεια. Παράδειγμα τέτοιας περιγραφής είναι τα πρότυπα του IEEE, τα οποία καθορίζουν τη δομή και το περιεχόμενο των παραγόμενων προϊόντων και στα οποία θα γίνει εκτενέστερη αναφορά σε επόμενο κεφάλαιο.



## ΕΝΟΤΗΤΑ 2.9. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ

### Δραστηριότητα 1/Κεφάλαιο 2

---

- Ένα μοντέλο κύκλου ζωής μπορεί να υλοποιείται στην πράξη από περισσότερες από μία διαδικασίες ανάπτυξης.
- Μια διαδικασία ανάπτυξης αφορά ακριβώς ένα μοντέλο κύκλου ζωής (συνήθως).
- Μια μεθοδολογία μπορεί να υποστηρίζεται από περισσότερα του ενός εργαλεία.
- Ένα εργαλείο μπορεί να υποστηρίζει περισσότερες από μία μεθοδολογίες.

Αν οι δικές σας απόψεις αποκλίνουν από τις παραπάνω διατυπώσεις, τότε απλά εμφανίζεται ένα φυσιολογικό για την πρώτη γνωριμία με το λογισμικό φαινόμενο. Μπορείτε να ξαναδιαβάσετε την Ενότητα 2.1 και ιδιαίτερα να μελετήσετε το Σχήμα 2.2. Όπως και να έχει, η ενασχόληση με πραγματικές ασκήσεις ανάπτυξης λογισμικού θα σας βοηθήσει να αποσαφηνήσετε στην πράξη όλες αυτές τις έννοιες και τις σχέσεις μεταξύ τους.

### Δραστηριότητα 2/Κεφάλαιο 2

---

1. Δεν είναι δυνατόν να υπάρχει μια πρώτη εικόνα του συστήματος λογισμικού που κατασκευάζεται παρά μόνο σε προχωρημένη φάση της ανάπτυξης.
2. Όσο μεγαλώνει η έκταση της εφαρμογής λογισμικού που κατασκευάζεται, τόσο δυσκολότερη γίνεται η μετάβαση από τη μία φάση στην επόμενη και η αποφυγή σφαλμάτων που δεν είναι δυνατό να εντοπιστούν παρά σε πολύ προχωρημένες φάσεις της ανάπτυξης.
3. Όσο αργότερα στην ανάπτυξη εντοπίζεται ένα σφάλμα, τόσο μεγαλύτερες είναι οι επιπτώσεις που η διόρθωσή του μπορεί να έχει σε όρους κόστους οπισθοδρομήσεων και επανάληψης

τουλάχιστον μέρους της διαδικασίας, παρενεργειών και γέννησης και νέων σφαλμάτων, καθυστερήσεων κ.λπ.

Οι παραπάνω διατυπώσεις (ιδιαίτερα οι 2 και 3) μπορούν να ακούγονται περισσότερο διασκεδαστικές –εξίσου εύστοχες– αν θυμηθούμε τον παραλληλισμό με τον ογκόλιθο της παραγράφου 2.1: «Όσο μεγαλύτερος ο ογκόλιθος, τόσο δυσκολότερη η μεταφορά του» και «Όσο αργότερα διαπιστώνεται ότι ακολουθούμε λάθος πορεία, τόσο περισσότερο δρόμο προς τα πίσω έχουμε να διανύσουμε μεταφέροντας μεγάλος βάρος, προκειμένου να ξαναβρούμε το σωστό δρόμο».

## Δραστηριότητα 3/Κεφάλαιο 2

---

Σκεφτείτε ότι κάθε κύκλος ανάπτυξης μπορεί να εννοηθεί ως επανάληψη, ως βήμα επαύξησης ή ως παράλληλη εκτέλεση τμήματος του έργου. Επίσης, σκεφτείτε ότι το πλάνο εργασιών μπορεί να συμπεριλαμβάνει την εκτίμηση του ρίσκου συνέχισης της ανάπτυξης.

## Άσκηση 1/Κεφάλαιο 2

---

Από τα όσα αναφέρονται στο Κεφάλαιο 1, σε περίπτωση που για οποιοδήποτε λόγο μεταβληθούν οι λειτουργικές απαιτήσεις από μια εφαρμογή λογισμικού κατά τη διάρκεια της ανάπτυξης αυτής σύμφωνα με το μοντέλο της λειτουργικής επαύξησης, μπορεί να καταστεί ακατάλληλη η κατάτμηση της εφαρμογής που έχει γίνει και να πρέπει να επαναληφθεί μεγάλο μέρος της ανάπτυξης. Σχετικά μπορείτε να ανατρέξετε και στις Ενότητες 1.4 και 1.5. Το λεπτό σημείο εδώ είναι η διατύπωση «ακατάλληλη η κατάτμηση της εφαρμογής». Αν δεν το εντοπίσατε αμέσως, αυτό είναι απολύτως φυσιολογικό, ιδιαίτερα για τους έχοντες μικρή πρακτική εμπειρία. Η Τεχνολογία Λογισμικού στοχεύει στην ελαχιστοποίηση της πιθανότητας να συμβεί κάτι τέτοιο.

## Άσκηση 2/Κεφάλαιο 2

---

Το κόστος για την εκτέλεση των ενεργειών προγραμματισμού εργασιών, εκτίμησης ρίσκου κ.λπ. κάθε άλλο παρά αμελητέο μπορεί να χαρακτηριστεί. Είναι αντιληπτό ότι το κόστος αυτό έχει μια ελάχιστη τιμή πέραν της οποίας δεν μπορεί να μειωθεί, όσο και να μειώνεται το μέγεθος της εφαρμογής λογισμικού που αναπτύσσεται. Στις περιπτώσεις μικρών εφαρμογών λογισμικού το κόστος αυτό είναι δυσανάλογο σχετικά με το καθαρό κόστος των ενεργειών ανάπτυξης, ως εκ τούτου μπορεί η χρήση του σπειροειδούς μοντέλου να μην αποτελεί την καλύτερη από οικονομικής άποψης επιλογή. Επιπροσθέτως, τα μειονεκτήματα άλλων μοντέλων κύκλου ζωής ελαχιστοποιούνται για μικρές εφαρμογές λογισμικού, οπότε μπορεί να εξεταστεί η χρήση ενός άλλου μοντέλου κύκλου ζωής.

## ΒΙΒΛΙΟΓΡΑΦΙΑ

Boehm, B. W., *A Spiral Model of Software Development and Enhancement*, IEEE Computer, May 1988, pp. 61-72, 1988.

Boehm, B. W., *Software life cycle factors*, *Handbook of Software Engineering*, edited by C. Vick and C. V. Ramamoorthy, Van Nostrand Reinhold, New York, pp. 494-518, 1984.

Davis, A. M., E. H. Bersoff and E. R. Comer, *A Strategy for Comparing Alternative Software Development Life Cycle Models*, IEEE Trans. on Soft. Eng., Vol. 14, No. 10, pp. 1453-1461, 1988.

Fairley, R. E., *Software Engineering Concepts*, McGraw-Hill, 1985.

Pressman, R. S., *Software Engineering-A Practitioners Approach*, McGraw-Hill.

Scacchi, W., *Models of Software Evolution: Life Cycle and Process*, SEI Curriculum Module SEI-CM-10-1.0, Carnegie Mellon University, Software Engineering Institute, 1987.

Sommerville, I. *Software Engineering*, London: Addison-Wesley.

Williams, R. D., *Management of Software Development*, *Handbook of Software Engineering*, Edited by C. R. Vick and C. V. Ramamoorthy, Van Nostrand Reinhold, 1984.

## ΑΡΧΙΤΕΚΤΟΝΙΚΕΣ ΔΙΑΤΑΞΕΙΣ ΛΟΓΙΣΜΙΚΟΥ

Σκοπός του κεφαλαίου είναι η εισαγωγή της έννοιας της διάταξης λογισμικού, ως αρχιτεκτονικής δόμησης των υπολογιστικών πόρων και της ανάθεσης σε αυτούς συστατικών στοιχείων λογισμικού. Ο προσδιορισμός των συστατικών στοιχείων λογισμικού που ανατίθενται (τοποθετούνται) για εκτέλεση σε υπολογιστικούς πόρους μπορεί να γίνει με διάφορες μεθοδολογίες οι σημαντικότερες από τις οποίες θα μας απασχολήσουν στη συνέχεια του βιβλίου. Ωστόσο η τοποθέτηση της αναφοράς στο αντικείμενο στο σημείο αυτό κρίνεται σκόπιμη διότι η βασική γνώση σχετικά με τις σύγχρονες αρχιτεκτονικές διατάξεις λογισμικού είναι χρήσιμη κατά τα επόμενα βήματα του προσδιορισμού των απαιτήσεων και της σχεδίασης του λογισμικού.

Μετά τη μελέτη του κεφαλαίου αυτού, ο αναγνώστης θα είναι σε θέση να:

- Ορίσει την έννοια της αρχιτεκτονικής διάταξης λογισμικού και να αναφέρει τέσσερις τέτοιες διατάξεις.
- Αντιλαμβάνεται τις δυνατότητες από την αξιοποίηση καταναμεμημένων αρχιτεκτονικών λογισμικού.

### Έννοιες-κλειδιά

---

- Διάταξη λογισμικού
- Πελάτης-εξυπηρετητής
- Τριμερής διάταξη λογισμικού
- Πολυμερής διάταξη λογισμικού

### Σύνοψη

---

Τα σύγχρονα υπολογιστικά περιβάλλοντα και οι συναφείς τεχνολογίες επιτρέπουν την κατάτμηση της λειτουργικότητας μιας εφαρμογής λογισμικού σε επιμέρους

δομικά τμήματα. Καθένα από αυτά δεν είναι απαραίτητο να εκτελείται στην ίδια υπολογιστική μηχανή, αλλά μπορεί να ανατίθεται σε ανεξάρτητους υπολογιστικούς πόρους οι οποίοι συνδέονται μέσω κάποιου δικτύου, ανταλλάσσουν δεδομένα και υπηρεσίες και τελικά όλο το σύστημα μαζί ικανοποιεί τις απαιτήσεις για τις οποίες κατασκευάστηκε. Με τον τρόπο αυτό μεταβαίνουμε από την ανάθεση όλης της λειτουργικότητας του λογισμικού σε μια μηχανή (μονολιθική διάταξη) στο διαμοιρασμό αυτής σε δύο (διάταξη *client-server*) ή περισσότερους υπολογιστικούς πόρους (τριμερείς και πολυμερείς διατάξεις).

## Εισαγωγικές παρατηρήσεις

---

Η εξάπλωση του «προσωπικού υπολογιστή» έφερε μια πραγματική επανάσταση στην αγορά η οποία έδωσε την αναγκαία ώθηση για να φτάσουμε στη σημερινή πραγματικότητα. Αρχικά, κάθε επιχείρηση απέκτησε τον «δικό της» υπολογιστή ο οποίος εκτελούσε συνήθως μαθηματικούς υπολογισμούς με ακρίβεια και ταχύτητα. Στη συνέχεια η έννοια του υπολογιστή σταδιακά σταμάτησε να ταυτίζεται με την πραγματοποίηση μαθηματικών υπολογισμών, οι οποίοι, εξάλλου, δεν ήταν απαραίτητοι σε όλους και να επεκτείνεται σημαντικά. Εφαρμογές όπως η επεξεργασία κειμένου έβαλαν τη γραφομηχανή στο ράφι και σιγά-σιγά η τυπογραφία, η σχεδίαση και άλλες εφαρμογές που μπορούσαν να κινήσουν την αγορά, απέκτησαν ως εργαλείο αυτό που σήμερα θεωρείται αυτονόητο: τον ηλεκτρονικό υπολογιστή.

Τα δίκτυα υπολογιστών, ιδιαίτερα όταν βγήκαν από το χώρο των στρατιωτικών εφαρμογών, έδωσαν μια νέα δυνατότητα στον «προσωπικό υπολογιστή»: να μοιράζεται τη δουλειά. Με τον τρόπο αυτό έγινε δυνατός ο διαχωρισμός της διαχείρισης των δεδομένων από τη χρήση τους και αναπτύχθηκε το μοντέλο *client-server* (ελληνική απόδοση: «πελάτη-εξυπηρετητή»).

Η πραγματική επανάσταση, ωστόσο, ήρθε με το διαδίκτυο. Το Internet προσέφερε την υποδομή τα πάντα (δηλαδή κάθε είδους εργασία) να μοιράζονται οπουδήποτε (δηλαδή σε υπολογιστικούς πόρους που βρίσκονται «κάπου» στο διαδίκτυο). Η δυνατότητα αυτή χρειάστηκε να επωαστεί για σχετικά μεγάλο χρονικό διάστημα μέχρι να φτάσουμε να μιλάμε για αυτό που σήμερα ονομάζουμε «cloud». Το «σύννεφο» είναι ένα πρακτικά απειροσύνολο υπολογιστικών πόρων η πραγματική υπόσταση των οποίων γίνεται αδιάφορη προς τους χρήστες τους, με τρόπο που ο καθένας



μπορεί να χρησιμοποιεί πόρους που δεν θα μπορούσε να αποκτήσει ποτέ ο ίδιος. Έτσι, μπορεί κανείς να έχει εικονικούς υπολογιστές, δίκτυα, αποθήκευση κ.ά., χωρίς να απασχολείται με τη φυσική υπόστασή τους και τους κάθε είδους περιορισμούς και απαιτήσεις αυτής. Αυτό μας επιτρέπει να μιλάμε για υπηρεσίες και όχι μόνο για εφαρμογές λογισμικού.

Γνώση υποδομής για τις υπηρεσίες αυτές είναι οι αρχιτεκτονικές διατάξεις λογισμικού.

## ΕΝΟΤΗΤΑ 3.1. Η ΕΝΝΟΙΑ ΤΗΣ ΔΙΑΤΑΞΗΣ ΛΟΓΙΣΜΙΚΟΥ

Ακολουθώντας θα εισάγουμε την έννοια της αρχιτεκτονικής διάταξης ή σκέτο διάταξης λογισμικού. Ο όρος «διάταξη» αποδίδει στην ελληνική το αγγλικό «deployment». Θα παρουσιαστούν τέσσερα μοντέλα διατάξεων: το μονολιθικό μοντέλο, το μοντέλο πελάτη-εξυπηρετητή (client-server), το τριμερές μοντέλο (3-tier), καθώς και ένα γενικευμένο μοντέλο πολλαπλής διάταξης (multi-tier).

### Διάταξη λογισμικού

Διάταξη λογισμικού (software deployment) είναι η κατάτμηση μιας εφαρμογής σε ανεξάρτητα λειτουργικά τμήματα και η ανάθεση αυτών σε διατιθέμενους υπολογιστικούς πόρους (συστήματα, επεξεργαστές).

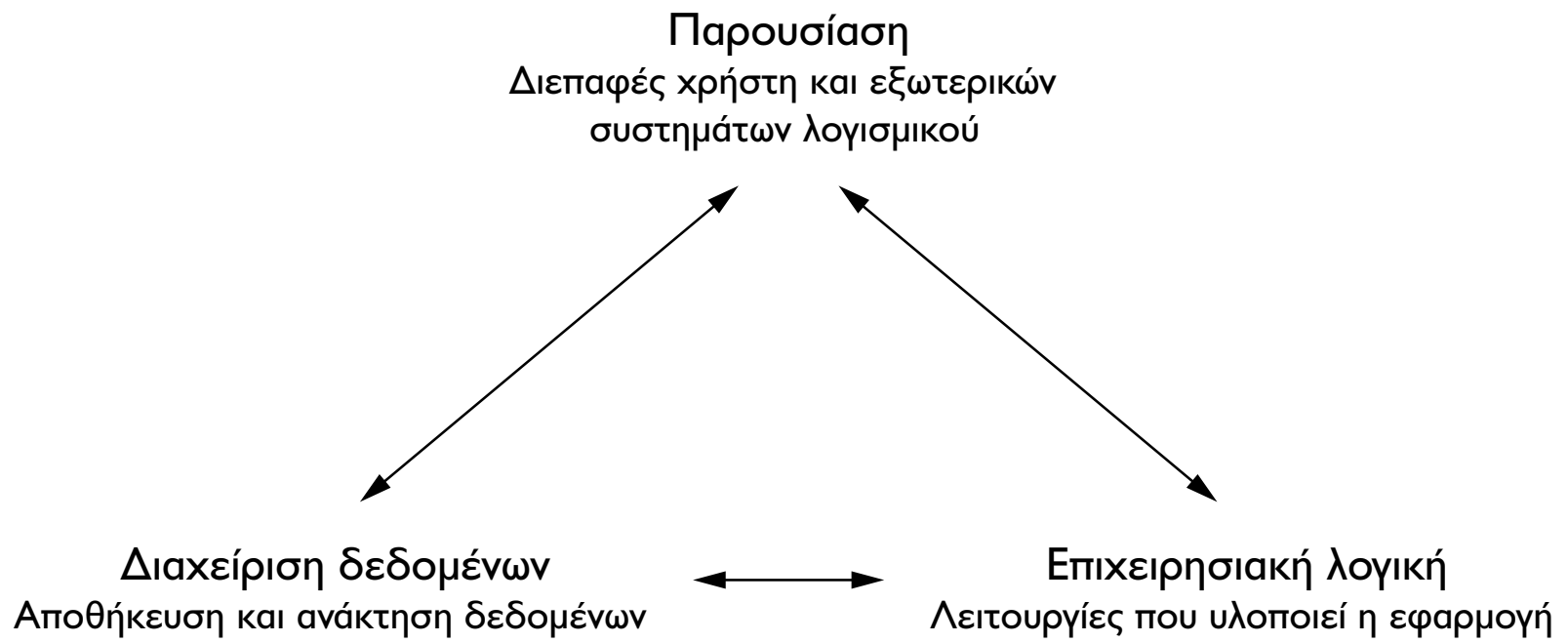
Με βάση τον παραπάνω ορισμό, με τον όρο «διάταξη» αναφερόμαστε στην γενική αρχιτεκτονική του λογισμικού. Σε αρκετές περιπτώσεις, χρησιμοποιείται σκέτος ο όρος «αρχιτεκτονική», ιδιαίτερα εκεί όπου το λογισμικό θεωρείται από εξωτερική σκοπιά και όχι από αυτή του κατασκευαστή. Ακολουθώντας θα αναφερόμαστε στη γενική αρχιτεκτονική με τον όρο «διάταξη», και στην εσωτερική αρχιτεκτονική, στην οποία ήδη αναφερθήκαμε, με τον όρο «αρχιτεκτονική».

Ο ορισμός των διατάξεων που θα ακολουθήσει θα βασιστεί στην κατηγοριοποίηση των εργασιών που μπορεί να κάνει μια εφαρμογή λογισμικού

όπως φαίνεται στο Σχήμα 3.1. Η κατηγοριοποίηση αυτή έχει γίνει γενικά αποδεκτή στην κοινότητα του λογισμικού. Διακρίνονται τρία είδη εργασιών: οι εργασίες παρουσίασης, οι εργασίες διαχείρισης δεδομένων και οι εργασίες επιχειρησιακής λογικής.



**Σχήμα 3.1** Μια διάκριση των εργασιών που κάνει μια εφαρμογή λογισμικού.



Ως εργασίες **παρουσίασης** ορίζονται όλες οι εργασίες που σχετίζονται με την επικοινωνία του συστήματος με το χρήστη και με εξωτερικές συσκευές και συστήματα, δηλαδή οι εργασίες που υλοποιούν τις διεπαφές του λογισμικού με το περιβάλλον του. Οι εργασίες **διαχείρισης δεδομένων** είναι εκείνες που ασχολούνται με την αποθήκευση και ανάκτηση των δεδομένων. Τέλος, οι εργασίες **επιχειρησιακής λογικής** (business logic) είναι όλες οι που υλοποιούν τις ιδιαίτερες λειτουργικές απαιτήσεις κάθε εφαρμογής λογισμικού.

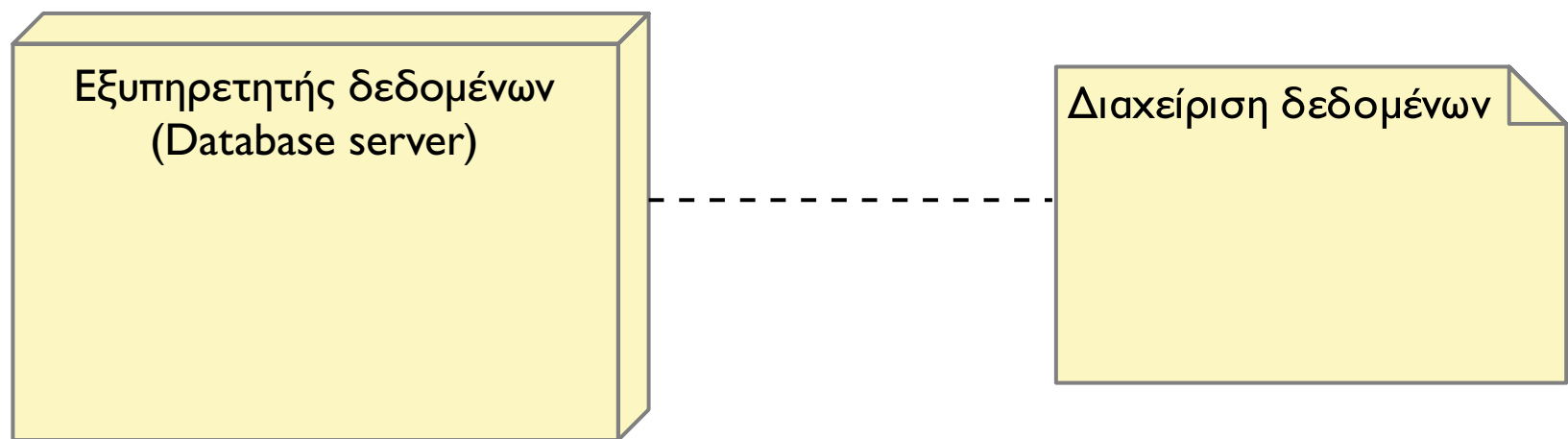
Η διάκριση αυτή απαιτεί μια αυστηρότητα στον ορισμό των μονάδων λογισμικού, η οποία είναι ιδιαίτερα σημαντική: Μια μονάδα λογισμικού η οποία εκτελεί έναν υπολογισμό (εργασία επιχειρησιακής λογικής) δε θα πρέπει να στέλνει η ίδια το αποτέλεσμα του σε καμία συσκευή εισόδου/εξόδου (διεπαφή, εργασία παρουσίασης). Αντίστοιχα, μια μονάδα που διαχειρίζεται δεδομένα δεν πρέπει να εκτελεί καμία υπολογιστική εργασία σε αυτά, όσο ελκυστική και αν φαίνεται μια τέτοια ιδέα κατά τη στιγμή του προγραμματισμού. Η πειθαρχία αυτή συχνά συγκρούεται με θέματα όπως ελαχιστοποίηση χρήσης μνήμης ή επιδόσεις. Ωστόσο, αποδίδει σχεδόν πάντα το κόστος της, καθώς κάνει ευκολότερη τη συντήρηση και επαναχρησιμοποίηση του λογισμικού.

## ΕΝΟΤΗΤΑ 3.2. ΤΥΠΙΚΕΣ ΔΙΑΤΑΞΕΙΣ ΛΟΓΙΣΜΙΚΟΥ

### 3.2.1. Η μονολιθική διάταξη

Η απλούστερη διάταξη λογισμικού είναι η μονολιθική (Σχήμα 3.2). Σε αυτήν, ολόκληρη η εφαρμογή τρέχει σε ένα και μόνο υπολογιστικό σύστημα. Η διάταξη αυτή είναι κατάλληλη για μικρές εφαρμογές με σχετικά περιορισμένες απαιτήσεις και λειτουργίες και, όπως είναι αναμενόμενο, υπήρξε η διάταξη που για μεγάλο χρονικό διάστημα χρησιμοποιήθηκε στους προσωπικούς υπολογιστές.

**Σχήμα 3.2** Η μονολιθική διάταξη λογισμικού.

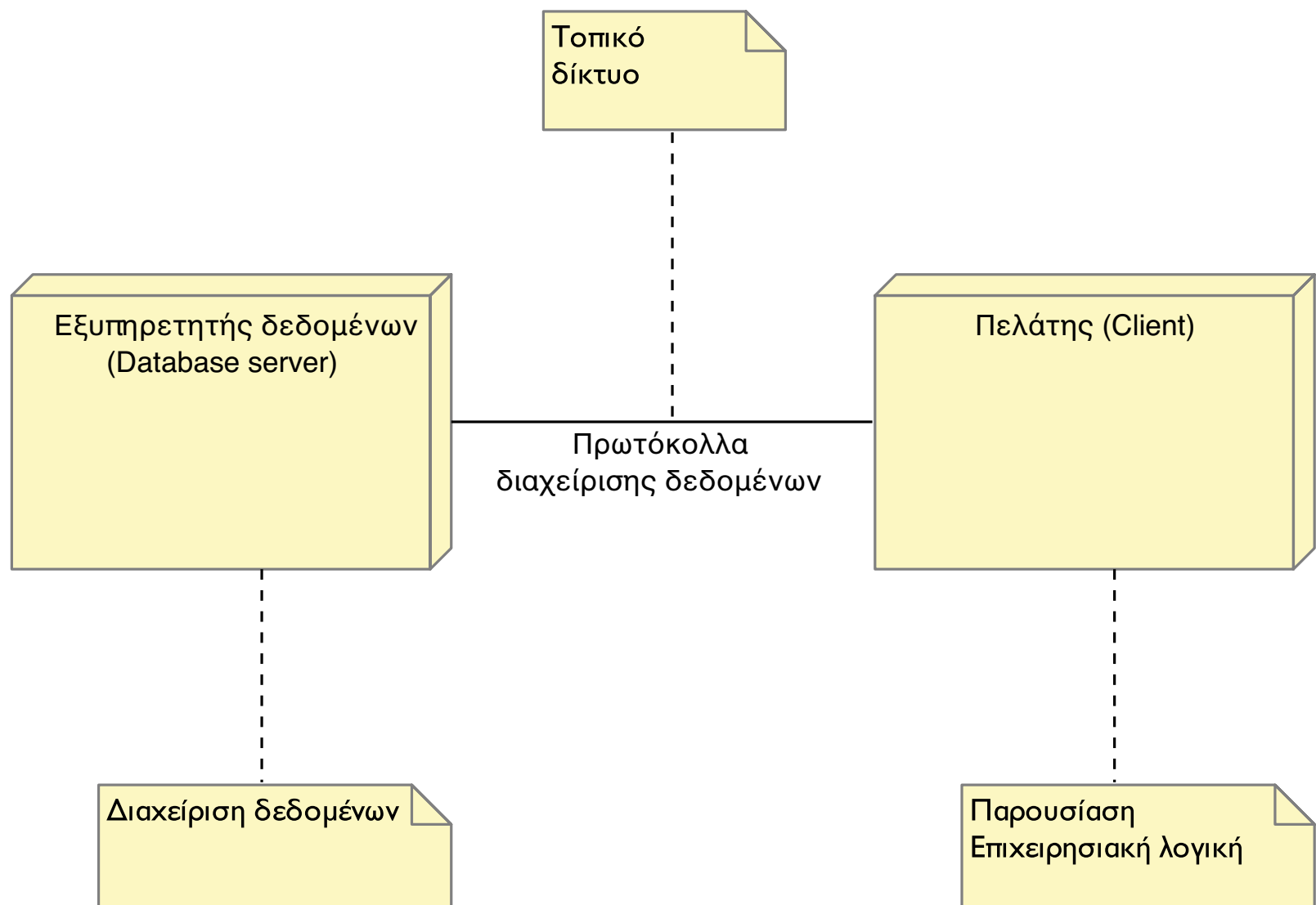


Ο συμβολισμός που χρησιμοποιείται στο 3.2 αναφέρεται ως «διάγραμμα διάταξης λογισμικού» (deployment diagram) και περιγράφει την ανάθεση τμημάτων της εφαρμογής σε υπολογιστικούς πόρους. Τα τμήματα αυτά συμβολίζονται με ένα τρισδιάστατο παραλληλεπίπεδο σκιασμένο όπως στο σχήμα, στην μπροστινή πλευρά του οποίου αναγράφεται η ονομασία κάθε τμήματος. Στο διάγραμμα διάταξης γίνεται αναλυτική αναφορά στην Ενότητα 9.2.

### **3.2.2. Η διάταξη πελάτη-εξυπηρετητή**

Η αύξηση των απαιτήσεων από το λογισμικό, της πολυπλοκότητας, αλλά και του όγκου των δεδομένων που διαχειρίζεται μια εφαρμογή, κατέστησαν την μονολιθική διάταξη ανεπαρκή για την ικανοποίηση πολλών απαιτήσεων. Παράλληλα, η ανάπτυξη των συστημάτων διαχείρισης σχεσιακών βάσεων δεδομένων, αλλά και των δικτύων, επέτρεψαν την εξέλιξη της μονολιθικής διάταξης σε αυτή του πελάτη-εξυπηρετητή (client-server).

**Σχήμα 3.3** Η διάταξη πελάτη-εξυπηρετητή.



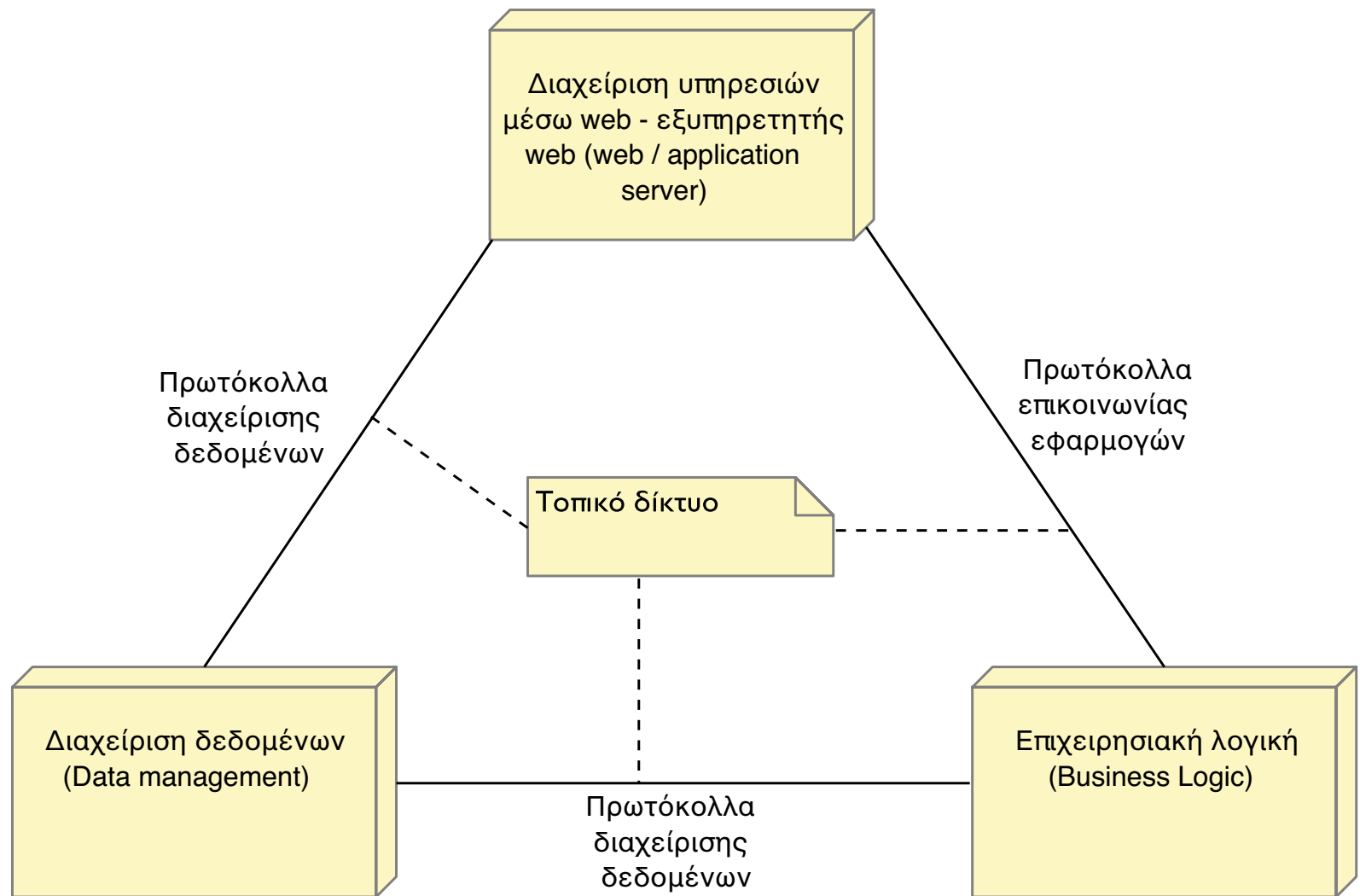
Στο Σχήμα 3.3 φαίνεται το διάγραμμα διάταξης πελάτη-εξυπηρετητή. Οι εργασίες που σχετίζονται με τη διαχείριση δεδομένων ανατίθενται σε ένα ξεχωριστό τμήμα της εφαρμογής, το οποίο τρέχει συνήθως σε ένα αφιερωμένο στη διαχείριση δεδομένων υπολογιστικό σύστημα. Οι εργασίες παρουσίασης και επιχειρησιακής λογικής τρέχουν σε ένα άλλο τμήμα, το οποίο επικοινωνεί μέσω δικτύου με τον εξυπηρετητή ζητώντας του την παροχή σχετικών με δεδομένα υπηρεσιών.

Η ιδέα, πρωτοποριακή για την εποχή της, έλυσε το πρόβλημα των ολοένα και μεγαλύτερων υπολογιστικών απαιτήσεων από τα γιγαντωμένα μονολιθικά συστήματα οι οποίες μεγάλωναν μαζί με τον αριθμό των χρηστών αλλά και την πολυπλοκότητα των εργασιών που εκτελούσαν. Με τον καιρό διάφορα προβλήματα της διάταξης αυτής αναδείχτηκαν. Το σημαντικότερο εντοπίζεται στην ανάγκη συντήρησης όλων των συστημάτων πελάτη (τα οποία μπορούσαν να είναι πολυάριθμα) καθώς συνέβαιναν οποιεσδήποτε μεταβολές στο επίπεδο της επιχειρησιακής λογικής. Επίσης, όσο μεγάλωνε η πολυπλοκότητα των λειτουργιών, τόσο περισσότερο τα συστήματα όπου έτρεχαν τα συστήματα πελάτη αποδεικνύονταν ανεπαρκή από πλευράς υπολογιστικής ισχύος.

### **3.2.3. Η τριμερής διάταξη**

Εμφανίστηκαν, λοιπόν, νέες εκδοχές της διάταξης πελάτη-εξυπηρετητή που διαχωρίζουν ακόμη περισσότερο τις λειτουργίες του λογισμικού. Το τμήμα λογισμικού «πελάτης» ελαφρύνεται και μένει μόνο με την ευθύνη της παρουσίασης, ενώ εμφανίζεται και ένας δεύτερος τύπος εξυπηρετητή, ο εξυπηρετητής εφαρμογών, ο οποίος κάνει τις εργασίες του επιπέδου της επιχειρησιακής λογικής. Η διάταξη αναφέρεται σαν «τριμερής» (3-tier) και εικονίζεται στο Σχήμα 3.4.

**Σχήμα 3.4** Η τριμερής διάταξη λογισμικού.



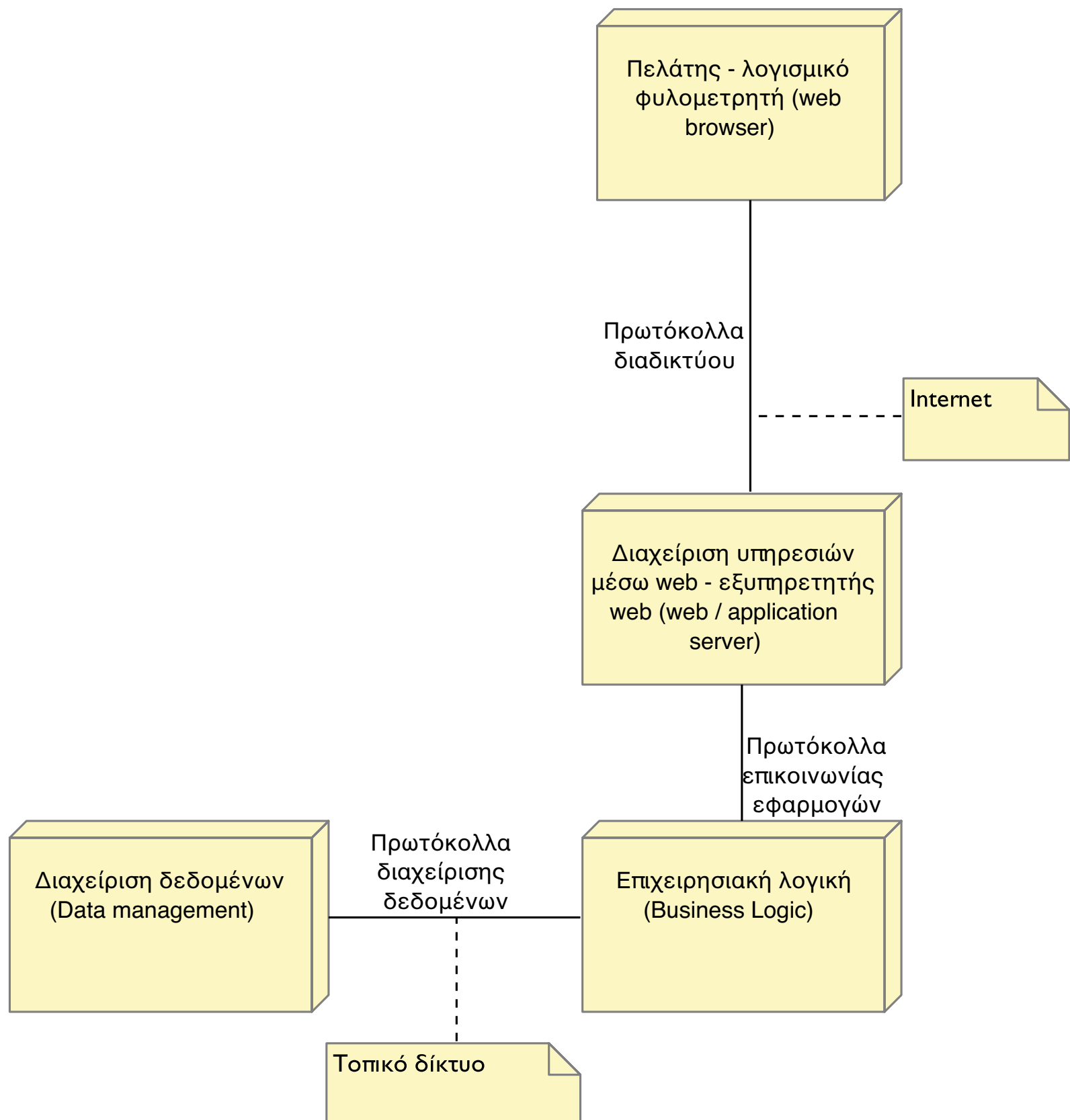
Στην περίπτωση αυτή, ο πελάτης χαρακτηρίζεται ως «ελαφρύς» (thin client) ακριβώς διότι κάνει λιγότερα πράγματα απ' ότι στην αρχική εκδοχή της διάταξης πελάτη-εξυπηρετητή. Τα συστήματα των εξυπηρετητών δεδομένων και εφαρμογών είναι συνήθως μεγάλα κεντρικά υπολογιστικά συστήματα. Τα προβλήματα της διάταξης πελάτη-εξυπηρετητή περιορίζονται, διότι οι απαιτήσεις συντήρησης των πελατών υφίστανται μόνον όταν συμβαίνουν μεταβολές στο επίπεδο της παρουσίας, χωρίς, πάντως, να σταματήσουν να υπάρχουν. Το σχήμα μπορεί να υλοποιηθεί πάνω από οποιοδήποτε δίκτυο, χωρίς τη χρήση πρωτοκόλλων τα οποία να είναι συμβατά με πρότυπα. Η ιδέα αποκτά μεγαλύτερο ενδιαφέρον όταν μεταφέρεται στο διαδίκτυο, είτε πραγματικά ως υλοποίηση, είτε χρησιμοποιώντας τεχνολογίες διαδικτύου μέσα σε τοπικό δίκτυο, όπως συζητείται στη συνέχεια.

#### **3.2.4. Η πολυμερής διάταξη**

Το επόμενο αναμενόμενο βήμα, είναι η αφαίρεση και της αρμοδιότητας της παρουσίας από τον πελάτη και η ανάθεσή της σε έναν εξυπηρετητή παρουσίας. Η ιδέα του εξυπηρετητή παρουσίας γεννήθηκε με την εμφάνιση του παγκοσμίου ιστού και του Internet και έγινε δυνατή με την ανάπτυξη τεχνολογιών που επιτρέπουν την αλληλεπίδραση μεταξύ του web server και του συνδεδεμένου σε αυτόν πελάτη (browser). Το πρόγραμμα πλοήγησης ή «φυλομετρητής» όπως έχει ατυχώς αποδοθεί στην ελληνική ο όρος «browser», έχει εξελιχθεί σε ένα πλήρες περιβάλλον στο οποίο μια εικονική μηχανή προσφέρει πολύ περισσότερες από υπηρεσίες παρουσίας, με την έννοια της απλής εμφάνισης περιεχομένου στην οθόνη του χρήστη. Σήμερα είναι δυνατή κάθε είδους αλληλεπίδραση του χρήστη με τον browser, με κάθε συσκευή που είναι διαθέσιμη στον υπολογιστή: πληκτρολόγιο, ποντίκι, κάμερα, κ.λπ. Η διάταξη ονομάστηκε πολυμερής (multi-tier) ή βασισμένη-στο-web (web based) και εμφανίζεται στο Σχήμα 3.5.



**Σχήμα 3.5** Μια πολυμερής διάταξη λογισμικού.

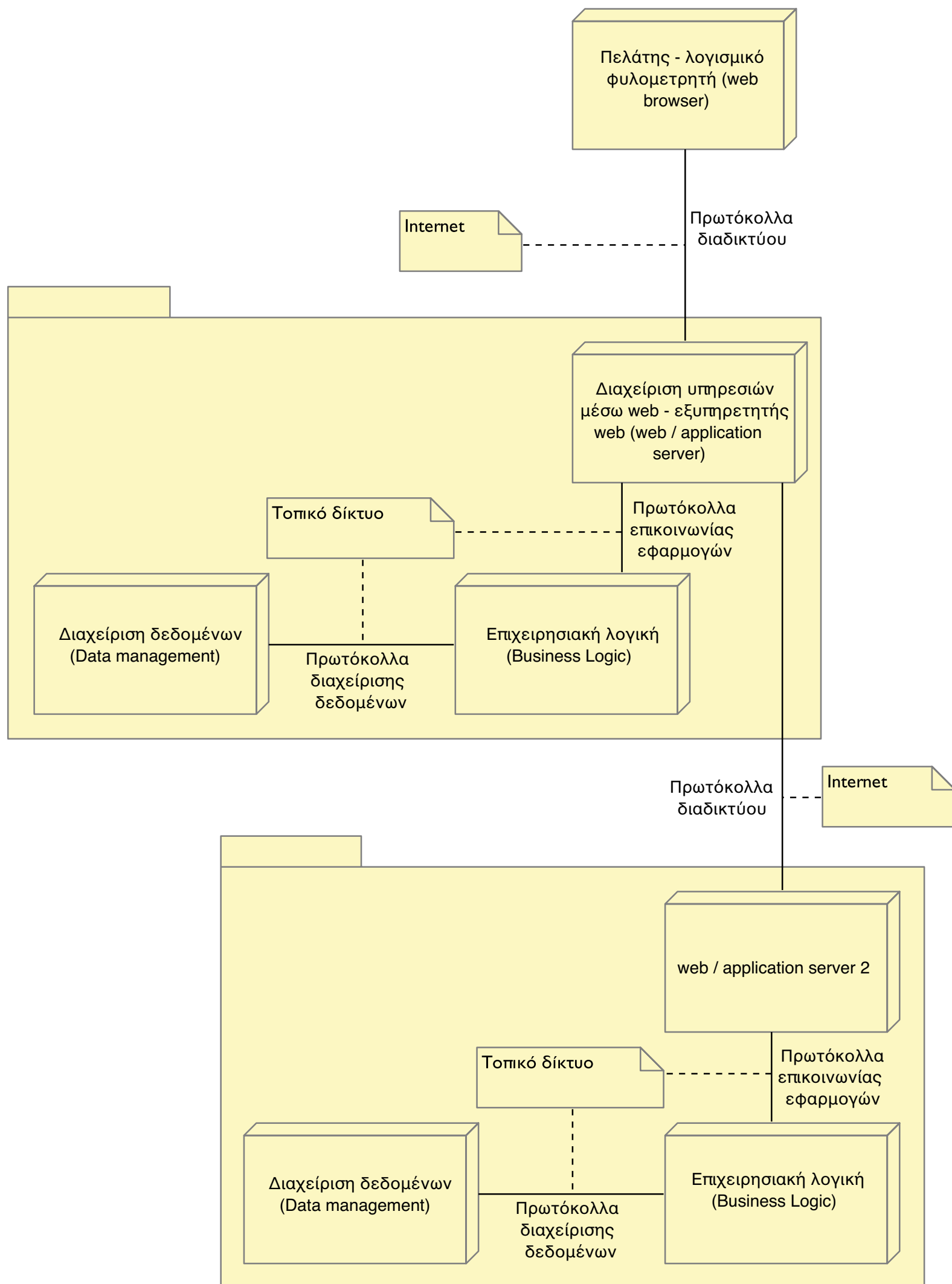


Τα τμήματα της εφαρμογής λογισμικού διατάσσονται με τρόπο που όλες οι εργασίες των τριών κατηγοριών (παρουσίασης, διαχείρισης δεδομένων και επιχειρησιακής λογικής) εκτελούνται σε έναν ή περισσότερους για κάθε κατηγορία εξυπηρετητές. Οι τεχνολογίες που χρησιμοποιούνται για την υλοποίηση τέτοιων αρχιτεκτονικών στο διαδίκτυο διακρίνονται σε αυτές που αφορούν την πλευρά του πελάτη (client-side), την πλευρά του εξυπηρετητή (server-side) και τα πρωτόκολλα παροχής υπηρεσιών πάνω από το web (web services). Ο εξυπηρετητής παρουσίασης δεν είναι παρά ένας εξυπηρετητής web (web server). Ο πελάτης δεν απαιτείται να διαθέτει κανένα τμήμα της εφαρμογής παρά μόνο τη δυνατότητα επικοινωνίας με τον web server, δηλαδή μια δικτυακή σύνδεση και ένα πρόγραμμα πλοήγησης στο web (browser). Για το λόγο αυτό ο πελάτης αναφέρεται και ως «web client». Τα προβλήματα ανάγκης συντήρησης των συστημάτων των πελατών εκμηδενίζονται, δημιουργούνται, όμως, άλλα, αυτά της ταχύτητας και της ασφάλειας των δικτυακών συνδέσεων, η αναφορά στα οποία είναι εκτός της εμβέλειας του παρόντος.

Η πολυμερής διάταξη εφαρμογών μπορεί να γίνει ιδιαίτερα σύνθετη: μία συγκεκριμένη εφαρμογή μπορεί να συντίθεται από τμήματα που βρίσκονται διάσπαρτα στο τοπικό δίκτυο ή το Internet και μάλιστα μπορούν περισσότερα του ενός τμήματα να προσφέρουν υπηρεσίες της ίδιας κατηγορίας. Για παράδειγμα, μπορούμε να έχουμε περισσότερους από έναν εξυπηρετητές διαχείρισης δεδομένων, επιχειρησιακής λογικής ή παρουσίασης. Επίσης, μια εφαρμογή μπορεί να λαμβάνει υπηρεσίες από άλλες εφαρμογές οι οποίες μπορούν με τη σειρά τους να είναι πολυμερείς ως προς την αρχιτεκτονική. Στην περίπτωση αυτή όλες οι υπηρεσίες που προσφέρονται πάνω από το διαδίκτυο τόσο αυτές που η μία εφαρμογή προσφέρει στην άλλη, όσο και αυτές που προσφέρονται στον τελικό πελάτη, χρησιμοποιούν δικτυακά πρωτόκολλα όπως το HTTP.

Μια τέτοια περίπτωση φαίνεται στο Σχήμα 3.6 που ακολουθεί.

**Σχήμα 3.6** Μια πολυμερής αρχιτεκτονική διάταξη λογισμικού πάνω από το διαδίκτυο.



## ΒΙΒΛΙΟΓΡΑΦΙΑ

Booch, G., *Object-Oriented Analysis and Design with Applications*, Addison-Wesley.

Martin J., Odell J., *Object-Oriented Analysis and Design*, Prentice Hall.

Pressman, R. S., *Software Engineering-A Practitioners Approach*, McGraw-Hill.

Sommerville, I. *Software Engineering*. London: Addison-Wesley.

## ΠΡΟΔΙΑΓΡΑΦΗ ΑΠΑΙΤΗΣΕΩΝ ΑΠΟ ΤΟ ΛΟΓΙΣΜΙΚΟ

Σκοπός του κεφαλαίου είναι ο ορισμός της έννοιας της απαίτησης από το λογισμικό και η παρουσίαση διαδικασιών προσδιορισμού και τρόπων περιγραφής τέτοιων απαιτήσεων, ακολουθώντας την προσέγγιση της δομημένης ανάλυσης.

Μετά τη μελέτη του κεφαλαίου αυτού, ο αναγνώστης θα είναι σε θέση:

- να διακρίνει τις απαιτήσεις από ένα σύστημα σε αυτές που αφορούν το σύστημα γενικά και σε εκείνες που αφορούν το λογισμικό ειδικότερα,
- να περιγράφει τουλάχιστον δύο λόγους για τους οποίους ο ορισμός απαιτήσεων από το λογισμικό είναι το πρώτο ιδιαίτερα σημαντικό βήμα κατά την ανάπτυξη,
- να χρησιμοποιεί τεχνικές δομημένης ανάλυσης, ώστε να ορίζει τις απαιτήσεις από μία εφαρμογή λογισμικού,
- να διακρίνει τις απαιτήσεις από το λογισμικό σε λειτουργικές και σε μη λειτουργικές,
- να περιγράφει τις απαιτήσεις αυτές με τη βοήθεια δομημένου κειμένου και διαγραμμάτων ροής δεδομένων, οντοτήτων – συσχετίσεων, μετάβασης καταστάσεων, καθώς και με τη χρήση ενός λεξικού δεδομένων,
- να διακρίνει τα γενικά χαρακτηριστικά μιας εφαρμογής λογισμικού μελετώντας τις απαιτήσεις από το λογισμικό,
- να δώσει τουλάχιστον δύο παραδείγματα απαιτήσεων από εφαρμογές λογισμικού με το αντικείμενο των οποίων έχει μια στοιχειώδη εξοικείωση και
- να αναφέρει τέσσερις κατηγορίες προβλημάτων που ανακύπτουν εξαιτίας της ελλιπούς εφαρμογής τεχνικών δομημένης ανάλυσης για τον ορισμό των απαιτήσεων από το λογισμικό.

- Μηχανική απαιτήσεων
- Απαίτηση από το σύστημα
- Απαίτηση από το λογισμικό
- Δομημένη ανάλυση
- Διάγραμμα ροής δεδομένων
- Διάγραμμα οντοτήτων – συσχετίσεων
- Διάγραμμα μετάβασης καταστάσεων
- Λεξικό δεδομένων

## Σύνοψη

---

Η προδιαγραφή των απαιτήσεων από το λογισμικό είναι η πιο δύσκολη και δημιουργική εργασία κατά την ανάπτυξη του λογισμικού. Απαιτεί ιδιαίτερες δυνατότητες επικοινωνίας, δομημένη και κριτική σκέψη και συστηματική προσέγγιση. Η Τεχνολογία Λογισμικού παρέχει το μεθοδολογικό πλαίσιο για την πραγματοποίηση της εργασίας αυτής και τα εργαλεία καταγραφής των αποτελεσμάτων της. Στο κεφάλαιο αυτό παρουσιάζεται η προσέγγιση που αναφέρεται ως δομημένη ανάλυση και τα αντίστοιχα μέσα καταγραφής των απαιτήσεων από το λογισμικό, δηλαδή το έγγραφο προδιαγραφών των απαιτήσεων και τα διαγράμματα ροής δεδομένων, οντοτήτων – συσχετίσεων, μετάβασης καταστάσεων, καθώς και το λεξικό δεδομένων. Πρόκειται για χρήσιμα μέσα τα οποία, σε συνδυασμό με τα κατάλληλα εργαλεία ανάπτυξης λογισμικού, μπορούν να βοηθήσουν τον κατασκευαστή να περιγράψει ικανοποιητικά το λογισμικό που κατασκευάζει. Παρά τη συστηματοποίηση που εισάγει η Τεχνολογία Λογισμικού στην προσέγγιση του προβλήματος του προσδιορισμού και της καταγραφής των απαιτήσεων από το λογισμικό, η διαδικασία παρουσιάζει ακόμα πολλά προβλήματα, που γενικά ταξινομούνται ως προβλήματα επικοινωνίας, προτύπων, γλώσσας αλλά και οικονομικά.

*Η πρώτη από τις διαδικασίες ανάπτυξης λογισμικού που αναφέρονται στο Κεφάλαιο 2 είναι η προδιαγραφή, η οποία περιγράφεται ως καθορισμός των εργασιών που θα επιτελεί το λογισμικό, καθώς και των περιορισμών και των παραδοχών που ισχύουν. Ανεξάρτητα από το μοντέλο κύκλου ζωής που ακολουθείται, η προδιαγραφή είναι πάντα η πρώτη διαδικασία κατά την ανάπτυξη λογισμικού από την οποία προκύπτει η επιθυμητή εικόνα ολόκληρου (αν αυτό είναι δυνατόν) ή έστω ενός τμήματος του λογισμικού που κατασκευάζεται. Ανάλογα με το μοντέλο κύκλου ζωής, η προδιαγραφή μπορεί να αναλύεται σε περισσότερες από μία φάσεις ή ακόμη και σε επιμέρους εργασίες. Σε κάθε περίπτωση, το αποτέλεσμα της διαδικασίας της προδιαγραφής είναι ο ορισμός των απαιτήσεων, έτσι ώστε να είναι δυνατή η κατασκευή του λογισμικού.*

*Όπως θα διαπιστώσει ο αναγνώστης, πρόκειται για την πιο κρίσιμη εργασία, την ανάπτυξη του λογισμικού, η οποία μπορεί να εξασφαλίσει την επιτυχία της ανάπτυξης αλλά και να θέσει τη βάση για την πλήρη αποτυχία αυτής. Εκτός από την πιο κρίσιμη, η προδιαγραφή των απαιτήσεων είναι, ίσως, η πιο δημιουργική εργασία κατά την ανάπτυξη του λογισμικού. Στο κεφάλαιο αυτό θα μελετήσουμε τη διαδικασία της προδιαγραφής των απαιτήσεων με χρήση της μεθοδολογίας της δομημένης ανάλυσης χωρίς να αναφερόμαστε σε κάποιο συγκεκριμένο μοντέλο κύκλου ζωής.*

## ΕΝΟΤΗΤΑ 4.1. Η ΕΝΝΟΙΑ ΤΗΣ ΑΠΑΙΤΗΣΗΣ ΑΠΟ ΤΟ ΛΟΓΙΣΜΙΚΟ

Στην ενότητα αυτή θα δοθεί ο ορισμός της απαίτησης από το σύστημα και το λογισμικό, καθώς και η ταξινόμηση των απαιτήσεων αυτών σε επιμέρους κατηγορίες.

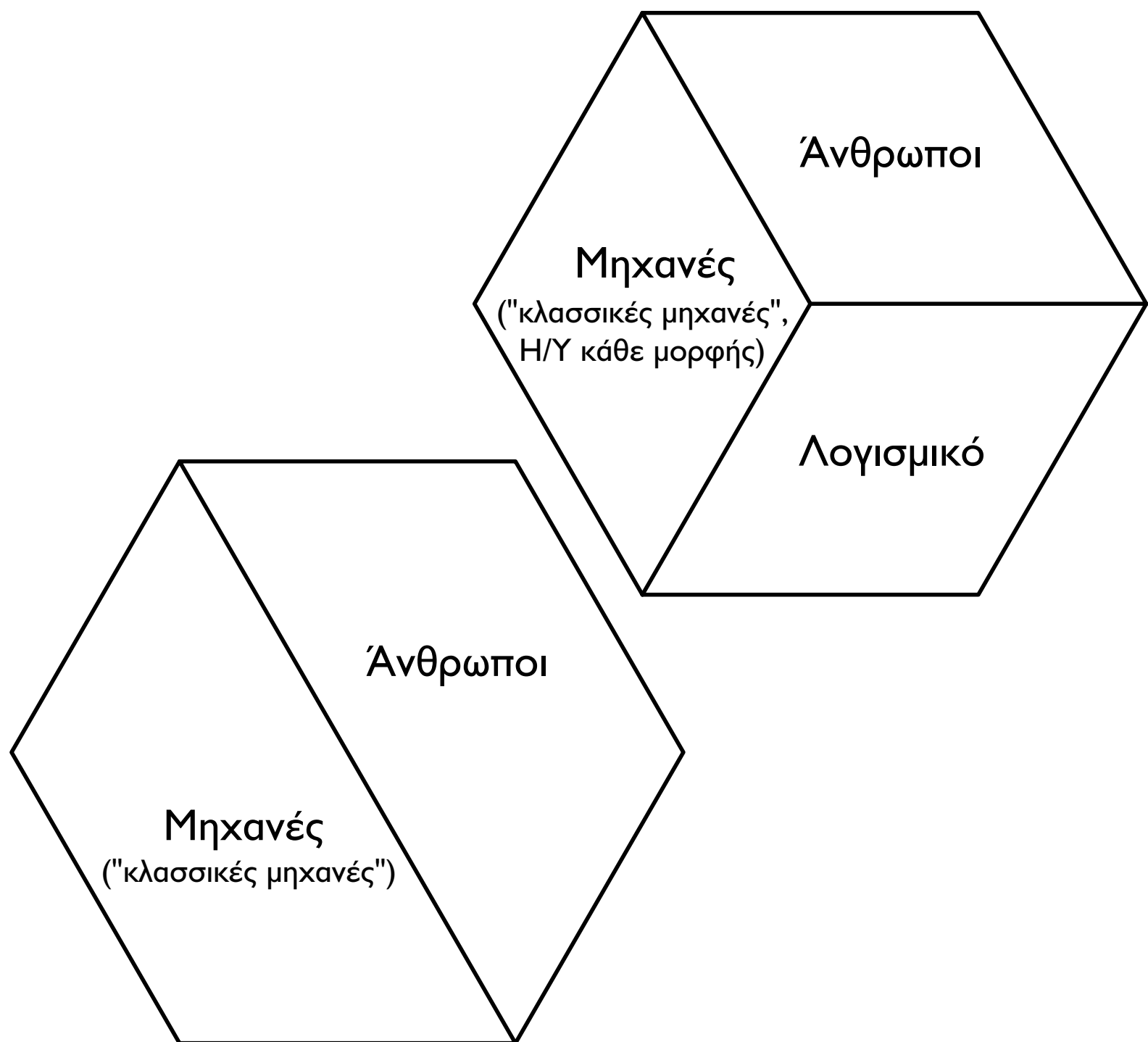
### 4.1.1. Απαιτήσεις από το σύστημα

Σε κάθε σύστημα του πραγματικού κόσμου μπορούμε να διακρίνουμε δύο συνιστώσες: τους ανθρώπους και τις μηχανές. Σύμφωνα με τη διάκριση αυτή, ο ηλεκτρονικός υπολογιστής είναι, ασφαλώς, μια μηχανή, η οποία όμως

παρουσιάζει σημαντικές διαφορές από τις μηχανές που ήταν γνωστές πριν από την εμφάνισή του. Συνοπτικά, οι διαφορές αυτές εστιάζονται στο ότι ο ηλεκτρονικός υπολογιστής δεν έχει υπόσταση παρά μόνο με τη βοήθεια του λογισμικού και, μάλιστα, η υπόσταση αυτή είναι διαφορετική ανάλογα με το λογισμικό που χρησιμοποιείται. Όταν, λοιπόν, σε ένα σύστημα συμπεριλαμβάνεται και ο ηλεκτρονικός υπολογιστής, είναι σκόπιμο να διακρίνουμε και μια τρίτη συνιστώσα, αυτή του λογισμικού (Σχήμα 4.1).



**Σχήμα 4.1** Μια διάκριση των συνιστωσών των συστημάτων πριν (αριστερά) και μετά (δεξιά) την εμφάνιση των Η/Υ και του λογισμικού.



Ο προσδιορισμός των απαιτήσεων από το σύστημα είναι μια εργασία παρόμοια με τον προσδιορισμό των απαιτήσεων από το λογισμικό, μόνο που, εκτός από το λογισμικό, αφορά και τις άλλες συνιστώσες ενός συστήματος, δηλαδή τους ανθρώπους και τις μηχανές. Κάτω από το πρίσμα αυτό, είναι μια ιδιαίτερα σύνθετη και συνάμα ενδιαφέρουσα εργασία, η οποία αποτελεί αυτοτελές γνωστικό αντικείμενο. Αρκετές από τις απαιτήσεις από το σύστημα μπορεί να σχετίζονται έμμεσα ή άμεσα με απαιτήσεις από το λογισμικό. Η διάκριση μεταξύ απαιτήσεων από το σύστημα και απαιτήσεων από το λογισμικό δεν είναι πάντα εύκολη και συχνά δημιουργεί σύγχυση, είναι ωστόσο χρήσιμη για τον σαφέστερο προσδιορισμό της υπόστασης του λογισμικού αλλά και για την καλύτερη κατανόηση ολόκληρου του συστήματος. Ένας απλός τρόπος να διακρίνουμε αν μια απαίτηση που διατυπώνεται αφορά ολόκληρο το σύστημα ή το λογισμικό είναι να προσπαθούμε να απαντήσουμε στην ερώτηση: «Ποια από τις συνιστώσες του συστήματος πρέπει να ικανοποιήσει την απαίτηση αυτή;». Αν η απάντηση «δείχνει» το λογισμικό, τότε μιλάμε για απαίτηση από το λογισμικό.

### **Απαίτηση από το σύστημα:**

Μια απαίτηση από το σύστημα είναι η περιγραφή μιας εργασίας που θα πρέπει να εκτελείται από κάποια εκ των συνιστωσών του συστήματος (άνθρωποι, μηχανές, λογισμικό) ή ενός χαρακτηριστικού το οποίο θα πρέπει να έχει ένα σύστημα.

Σύμφωνα με τον ορισμό αυτό, οι απαιτήσεις από το λογισμικό είναι στην ουσία απαιτήσεις από το σύστημα. Θα πρέπει, ωστόσο, να σημειωθεί ότι κατά τον προσδιορισμό των απαιτήσεων από το σύστημα η προσοχή είναι εστιασμένη στην κατασκευή ολόκληρου του συστήματος και ότι η δομή και η λεπτομέρεια με τις οποίες γίνεται η περιγραφή των απαιτήσεων που αφορούν το λογισμικό δεν είναι επαρκείς για την κατασκευή του. Οι απαιτήσεις από το σύστημα μπορούν να καταγράφονται με διάφορους τρόπους, ο πιο απλός εκ των οποίων είναι το απλό κείμενο, συνοδευόμενο ενδεχομένως από σχήματα. Περισσότερο δομημένοι τρόποι περιγραφής έχουν

προταθεί με διάφορα πρότυπα, όπως αυτά του IEEE, που αναφέρονται στη βιβλιογραφία.

### **Παράδειγμα I/Κεφάλαιο 4**

Ζητείται η κατασκευή ενός συστήματος παρακολούθησης μετεωρολογικών μετρήσεων το οποίο με χρήση ειδικών αισθητηρίων οργάνων συλλέγει από διάφορα γεωγραφικά σημεία δεδομένα θερμοκρασίας, ατμοσφαιρικής πίεσης και υγρασίας. Το σύστημα αποθηκεύει τα στοιχεία αυτά και, κατόπιν, εξάγει στατιστικά αποτελέσματα, όπως μέση τιμή και τυπική απόκλιση για κάθε γεωγραφικό σημείο. Το σύστημα αποτελείται από συσκευές μέτρησης (αισθητήρες) πίεσης, θερμοκρασίας και υγρασίας, από ηλεκτρονικούς υπολογιστές και από ανθρώπους.

Η περιγραφή των απαιτήσεων από το σύστημα μπορεί να περιγραφεί ως ακολούθως: το σύστημα εκτελεί την εργασία συλλογής μετεωρολογικών δεδομένων με τη βοήθεια ειδικών αισθητήρων, την ηλεκτρονική μετάδοση των δεδομένων αυτών σε κάποιο κεντρικό σταθμό, την αποθήκευση αυτών σε ηλεκτρονική μορφή, την επεξεργασία με την εξαγωγή χαρακτηριστικών μεγεθών, καθώς και την αξιολόγηση των μετρήσεων.

#### **4.1.2. Τι είναι «απαίτηση από το λογισμικό»;**

Όταν κατασκευάζουμε λογισμικό, το πρώτο που πρέπει να συλλάβουμε με όσο το δυνατό μεγαλύτερη σαφήνεια είναι οι εργασίες που αυτό θα πρέπει να κάνει, καθώς και άλλα χαρακτηριστικά που είναι επιθυμητό να έχει όπως, για παράδειγμα, η εμφάνιση, οι επιδόσεις, ο τρόπος χρήσης, η ασφάλεια κ.ά. Τόσο οι εργασίες, όσο και τα χαρακτηριστικά αυτά θα καθορίσουν σε σημαντικό βαθμό τις δραστηριότητες που θα ακολουθήσουν κατά την ανάπτυξη του λογισμικού. Είναι φανερό ότι η λανθασμένη ή έστω αποκλίνουσα αντίληψη των απαιτούμενων από το λογισμικό εργασιών και χαρακτηριστικών

μπορεί να οδηγήσει στην κατασκευή λογισμικού που δεν επιτελεί τον σκοπό του. Ως εκ τούτου, ο προσδιορισμός και η σαφής περιγραφή των απαιτήσεων είναι ένας ιδιαίτερα σημαντικός κρίκος στην αλυσίδα των εργασιών που εκτελούνται κατά τον κύκλο ζωής μιας εφαρμογής λογισμικού.

### **Απαίτηση από το λογισμικό:**

Μια απαίτηση από το λογισμικό είναι μια λειτουργία που αυτό θα πρέπει να επιτελεί ή μια συνθήκη που θα πρέπει να ικανοποιεί όταν θα έχει ολοκληρωθεί η κατασκευή του. .

Ακόμη και από τη θέση του χρήστη λογισμικού δεν είναι δύσκολο να αντιληφθεί κανείς ότι οι απαιτήσεις, έστω και από μια μικρή εφαρμογή λογισμικού, είναι πολλές και διαφορετικού χαρακτήρα, ενώ μπορούν να περιγραφούν με μικρότερη ή μεγαλύτερη λεπτομέρεια. Συνήθως ο πελάτης εκφράζει τις απαιτήσεις του με μεγάλο βαθμό γενικότητας, ενώ ο κατασκευαστής τις διατυπώνει με μεγαλύτερη λεπτομέρεια και σαφήνεια, ώστε να μπορεί να κάνει τη δουλειά του. Ας δούμε ένα απλό παράδειγμα.

## Παράδειγμα 2/Κεφάλαιο 4

Ζητείται η κατασκευή μιας εφαρμογής συντάκτη αρχείων κειμένου (text editor). Πρόκειται για μια σχετικά απλή εφαρμογή, εμπειρία σχετικά με την οποία έχουν σχεδόν όλοι οι χρήστες υπολογιστών. Οι απαιτήσεις από την εφαρμογή αυτή, όπως εκφράζονται από τον πελάτη, είναι οι ακόλουθες:

- Π1 Ο χρήστης θα πρέπει να πληκτρολογεί κείμενο έχοντας στη διάθεσή του όλες τις βασικές λειτουργίες που παρέχουν για το σκοπό αυτό τα Windows 98.
- Π2 Το κείμενο θα πρέπει να αποθηκεύεται με τη μορφή αρχείου στο δίσκο, καθώς και να ανακτάται από τον δίσκο για επεξεργασία.
- Π3 Θα πρέπει να παρέχεται μια μπάρα εργαλείων με τις εργασίες που εκτελούνται συχνότερα.

Για τον κατασκευαστή λογισμικού, μια πιο λεπτομερής διατύπωση των παραπάνω απαιτήσεων είναι η ακόλουθη:

- K1 Η εφαρμογή θα λειτουργεί σε περιβάλλον Windows.
- K2 Με την εκκίνηση της εφαρμογής, αυτή βρίσκεται σε κατάσταση συγγραφής νέου κειμένου.
- K3 Κατά τη λειτουργία συγγραφής κειμένου εμφανίζεται στην οθόνη, στην τρέχουσα θέση του δρομέα, ο χαρακτήρας που αντιστοιχεί στο πλήκτρο που πατά ο χρήστης.
- K4 Η θέση του δρομέα μπορεί να αλλάξει πατώντας τα βελάκια ή χρησιμοποιώντας το ποντίκι.
- K5 Το τράβηγμα (drag) του ποντικιού από ένα σημείο του κειμένου σε ένα άλλο έχει ως αποτέλεσμα τη σημείωση του ενδιαμέσου κειμένου ως επιλεγμένου και την εμφάνισή του με αντίστροφα χρώματα.

- K6 Το κείμενο παραμένει επιλεγμένο μέχρι να πατηθεί οποιοδήποτε πλήκτρο. Αν το πρώτο πλήκτρο που πατηθεί είναι αλφαριθμητικό, τότε το επιλεγμένο κείμενο διαγράφεται, ενώ, αν είναι βελάκι κατεύθυνσης, τότε η επιλογή αναιρείται χωρίς διαγραφή του επιλεγμένου κειμένου.
- K7 Με τα πλήκτρα **Del** και **Backspace** διαγράφεται ο χαρακτήρας που βρίσκεται δεξιά ή αριστερά του δρομέα αντίστοιχα ή ολόκληρο το επιλεγμένο κείμενο, αν υπάρχει ενεργή επιλογή κειμένου.
- K8 Η χρήση του πλήκτρου **Insert** αλλάζει την κατάσταση συγγραφής εναλλάσσοντάς τη μεταξύ επανογραφής και εισαγωγής κειμένου.
- K9 Οι συνδυασμοί των πλήκτρων **Control-C**, **Control-X** και **Control-V** έχουν το αποτέλεσμα της εκτέλεσης των λειτουργιών **Αντιγραφή**, **Αποκοπή** και **Επικόλληση** για το επιλεγμένο κείμενο.
- K10 Η εφαρμογή διαθέτει μενού με τις εντολές **Νέο αρχείο**, **Άνοιγμα**, **Αποθήκευση**, **Αποθήκευση ως** και **Έξοδος**.
- K11 Η εφαρμογή διαθέτει μια μπάρα εργαλείων που περιέχει όλες τις εντολές που αναφέρονται στο μενού, καθώς επίσης και τις εντολές **Αντιγραφή**, **Αποκοπή** και **Επικόλληση**.
- K12 Με την εκτέλεση της εντολής **Αποθήκευση ως** εμφανίζεται ένα παράθυρο διαλόγου απ' όπου επιλέγεται η τοποθεσία της αποθήκευσης και το όνομα, και ακολούθως γίνεται το γράψιμο του αρχείου στο δίσκο.

- KI3 Η εντολή **Αποθήκευση** έχει ως αποτέλεσμα το γράψιμο του αρχείου στο δίσκο. Αν πρόκειται για νέο αρχείο, εκτελείται η εντολή **Αποθήκευση** ως.
- KI4 Η εντολή **Άνοιγμα** εμφανίζει ένα παράθυρο διαλόγου απ' όπου επιλέγεται η τοποθεσία και το όνομα, και ακολούθως το αρχείο έρχεται από το δίσκο στη μνήμη και η εφαρμογή περνά σε κατάσταση συγγραφής.

Από το παράδειγμα, το οποίο δεν είναι πλήρες και εξυπηρετεί εισαγωγικούς σκοπούς, μπορούν να διατυπωθούν δύο πρώτες θέσεις οι οποίες θα τεκμηριωθούν περαιτέρω στη συνέχεια. Πρώτο, η περιγραφή των απαιτήσεων από το λογισμικό είναι διαφορετική για τον πελάτη απ' ό,τι για τον κατασκευαστή. Ο τελευταίος απαιτεί διατύπωση με μεγαλύτερη εκλέπτυνση και σαφήνεια, προκειμένου να υλοποιήσει το επιθυμητό λογισμικό. Ωστόσο, όχι σπάνια, οι απαιτήσεις του πελάτη μπορούν να ερμηνευτούν με περισσότερους του ενός τρόπους, γεγονός που δημιουργεί προβλήματα, και με το οποίο θα ασχοληθούμε σε επόμενη ενότητα.

Δεύτερο, δεν περιγράφουν όλες οι απαιτήσεις λειτουργίες που θα πρέπει να επιτελεί το λογισμικό. Κάποιες από αυτές περιγράφουν επιθυμητά χαρακτηριστικά τα οποία δεν σχετίζονται με λειτουργίες. Στο προηγούμενο παράδειγμα η απαίτηση KI δεν περιγράφει κάτι που πρέπει να κάνει το λογισμικό, αλλά ένα χαρακτηριστικό του και, συγκεκριμένα, το λειτουργικό σύστημα στο οποίο αυτό θα τρέχει, ενώ η απαίτηση K3 περιγράφει μια συγκεκριμένη λειτουργία που θα πρέπει να εκτελεί το λογισμικό. Αποκτά, λοιπόν, νόημα η ταξινόμηση των απαιτήσεων από το λογισμικό σε κατηγορίες.

#### 4.1.3. Πώς ταξινομούνται οι απαιτήσεις από το λογισμικό;

Οι απαιτήσεις από το λογισμικό διακρίνονται σε δύο μεγάλες κατηγορίες. Στις λειτουργικές και στις μη λειτουργικές.



### Λειτουργικές απαιτήσεις:

Οι λειτουργικές απαιτήσεις περιγράφουν τις εργασίες (λειτουργίες) που θα πρέπει να εκτελεί το λογισμικό.

Οι λειτουργικές απαιτήσεις καθορίζουν πλήρως τη συμπεριφορά του συστήματος, δηλαδή τα επιθυμητά αποτελέσματα που αυτό πρέπει να παράγει ή γενικά την απόκριση που πρέπει να εμφανίζει στο περιβάλλον του όταν ισχύουν συγκεκριμένες συνθήκες.

### Μη λειτουργικές απαιτήσεις:

Οι μη λειτουργικές απαιτήσεις περιγράφουν χαρακτηριστικά που πρέπει να έχει το λογισμικό, τα οποία δεν αφορούν την εκτέλεση κάποιας λειτουργίας από αυτό.

Οι μη λειτουργικές απαιτήσεις καθορίζουν ιδιώματα εμφάνισης, περιβάλλοντος λειτουργίας, επιδόσεων κ.ά., τα οποία γενικά χαρακτηρίζουν το λογισμικό, χωρίς όμως να μπορούν να ιδωθούν ως λειτουργίες που αυτό επιτελεί. Ως μη λειτουργικές χαρακτηρίζονται και οι απαιτήσεις που αφορούν κάποια από τις επόμενες φάσεις του κύκλου ζωής του λογισμικού. Οι μη λειτουργικές απαιτήσεις μπορούν να ταξινομηθούν σε επιμέρους κατηγορίες, οι οποίες αναφέρονται ακολούθως.

**Απαιτήσεις χρήσης:** Καθορίζουν τα χαρακτηριστικά της χρήσης του συστήματος, την αισθητική της επικοινωνίας με το χρήστη (user interface), καθώς και το υλικό τεκμηρίωσης και εκπαίδευσης που θα έχει στη διάθεσή του ο τελικός χρήστης. Παράδειγμα: *Το λογισμικό θα πρέπει να ελέγχεται με τη χρήση του ποντικιού ή του πληκτρολογίου και να συνοδεύεται από αναλυτικό εγχειρίδιο χρήστη και εγχειρίδιο εκμάθησης.*

**Απαιτήσεις αξιοπιστίας:** Καθορίζουν τη συμπεριφορά του λογισμικού σε καταστάσεις ενδογενών ή εξωγενών σφαλμάτων, τη διαδικασία αποκατάστασης, την πρόβλεψη τέτοιων καταστάσεων, καθώς και την επιθυμητή διαθεσιμότητα του λογισμικού. Παράδειγμα: *Σε περίπτωση απρόβλεπτου τερματισμού*



της λειτουργίας του λογισμικού θα πρέπει να επιχειρείται επανεκκίνηση με την ελάχιστη δυνατή απώλεια δεδομένων για το χρήστη.

**Απαιτήσεις επιδόσεων:** Εισάγουν περιορισμούς σε λειτουργικές απαιτήσεις σχετικά με τον χρόνο εκτέλεσής τους και με τη χρήση πόρων, όπως η μνήμη και οι μονάδες επεξεργασίας. Παράδειγμα: Ο χρόνος αναζήτησης και ανάκτησης από τη βάση δεδομένων μιας εγγραφής με κλειδί το ονοματεπώνυμο δεν θα πρέπει να ξεπερνά το ένα δευτερόλεπτο.

**Απαιτήσεις υποστήριξης:** Καθορίζουν τα επιθυμητά χαρακτηριστικά για τον έλεγχο και τη συντήρηση του λογισμικού. Παράδειγμα: Κατά την εγκατάσταση θα πρέπει να καταγράφεται σε αρχείο μη ορατό από τον χρήστη η έκδοση όλων των αρχείων που εγκαταστάθηκαν.

**Απαιτήσεις σχεδίασης:** Καθορίζουν τον τρόπο με τον οποίο θα πρέπει να γίνει η σχεδίαση του λογισμικού. Παράδειγμα: Η σχεδίαση θα πρέπει να γίνει με χρήση της μεθοδολογίας OMT και με χρήση του προτύπου IEEE Std 1016.

**Απαιτήσεις υλοποίησης:** Καθορίζουν τον τρόπο με τον οποίο θα πρέπει να γίνει η συγγραφή του πηγαίου κώδικα (source code) του λογισμικού. Παράδειγμα: Θα πρέπει να χρησιμοποιηθεί η γλώσσα ANSI C και να θεωρείται ότι η συνολική διαθέσιμη μνήμη είναι 64 KB.

**Απαιτήσεις επικοινωνίας με άλλα συστήματα:** Καθορίζουν τα εξωτερικά συστήματα λογισμικού ή άλλα συστήματα με τα οποία το λογισμικό θα επικοινωνεί, καθώς και τον τρόπο (λ.χ. πρότυπα, φυσική σύνδεση) πραγματοποίησης της επικοινωνίας αυτής. Παράδειγμα: Το λογισμικό θα επικοινωνεί με ένα σύστημα διαχείρισης βάσεων δεδομένων μέσω του πρωτοκόλλου ODBC.

**Απαιτήσεις βάσεων δεδομένων:** Καθορίζουν τις οντότητες για τη διαχείριση των οποίων είναι υπεύθυνο το σύστημα λογισμικού, καθώς και τα ιδιώματα καθεμίας από αυτές, όπως αυτά είναι αναγνωρίσιμα στην παρούσα φάση της ανάπτυξης. Παράδειγμα: Το λογισμικό θα πρέπει να διατηρεί αρχείο πελατών με τα εξής στοιχεία: ονοματεπώνυμο, διεύθυνση, τηλέφωνο, ΑΦΜ.

**Φυσικές απαιτήσεις:** Καθορίζουν τα επιθυμητά φυσικά χαρακτηριστικά του λογισμικού και του συστήματος. Παράδειγμα: Καθορισμός λειτουργικού συστήματος και προδιαγραφές υπολογιστή όπου θα τρέχει το λογισμικό, περιγραφή απαιτούμενων δικτυακών συνδέσεων.

Η ταξινόμηση που μόλις αναφέρθηκε φαίνεται παραστατικά στο Σχήμα 4.2. Αξίζει να σημειωθεί ότι υπάρχει στη βιβλιογραφία μεγάλο πλήθος ταξινομήσεων μη λειτουργικών απαιτήσεων.

**Σχήμα 4.2** Ταξινόμηση των απαιτήσεων από το λογισμικό.

## Απαιτήσεις από το λογισμικό

### Λειτουργικές

Λειτουργία 1

Λειτουργία 2

...

Λειτουργία N

### Μη-λειτουργικές

Χρήσης

επιθυμητά περιβάλλοντα και χαρακτηριστικά

Αξιοπιστίας

μέτρα αξιοπιστίας

Επιδόσεων

μέτρα επιδόσεων

Υποστήριξης

Σχεδίασης

απαιτήσεις συμμόρφωσης με  
αρχιτεκτονικές κλπ

Υλοποίησης

περιορισμοί από το περιβάλλον  
λειτουργίας και ανάπτυξης

Επικοινωνίας

συνεργασία με άλλα συστήματα,  
συσκευές, λογισμικά

Βάσεων δεδομένων

απαιτήσεις οργάνωσης δεδομένων

Φυσικές

απαιτήσεις συσκευών και  
περιβάλλοντος λειτουργίας

## Άσκηση 1/Κεφάλαιο 4

Προσπαθήστε να κατατάξετε τις απαιτήσεις του παραδείγματος του συντάκτη κειμένων (παράγραφος 4.1.2) στις κατηγορίες που αναφέρονται στην παράγραφο 4.1.3.

## Δραστηριότητα 1/Κεφάλαιο 4

Ορίστε τουλάχιστον τέσσερις λειτουργικές και δύο μη λειτουργικές απαιτήσεις από μια εφαρμογή λογισμικού που υλοποιεί ένας μικρός υπολογιστής τσέπης ο οποίος μπορεί να εκτελεί μόνο τις πράξεις πρόσθεση, αφαίρεση, πολλαπλασιασμό και διαίρεση.

## Δραστηριότητα 2/Κεφάλαιο 4

Ακολουθώς παραθέτουμε απόσπασμα από το παράδειγμα της παραγράφου 4.1.1 το οποίο αναφέρεται στις απαιτήσεις από ένα σύστημα συλλογής και επεξεργασίας μετεωρολογικών μεγεθών.

*...Το σύστημα αποθηκεύει τα στοιχεία αυτά και, κατόπιν, εξάγει στατιστικά αποτελέσματα, όπως μέση τιμή και τυπική απόκλιση για κάθε γεωγραφικό σημείο. Το σύστημα αποτελείται από συσκευές μέτρησης (αισθητήρες) πίεσης, θερμοκρασίας και υγρασίας, από ηλεκτρονικούς υπολογιστές και από ανθρώπους. Η περιγραφή των απαιτήσεων από το σύστημα περιλαμβάνει τη συλλογή, την αποθήκευση, την επεξεργασία και την αξιολόγηση των μετρήσεων.*

Περιγράψτε τουλάχιστον τρεις απαιτήσεις από το σύστημα και τρεις απαιτήσεις από το λογισμικό. Πώς εξειδικεύονται οι συνιστώσες του συστήματος και για ποιες εργασίες είναι υπεύθυνη καθεμία από αυτές;

## Άσκηση 2/Κεφάλαιο 4

**Ακολουθώς παρατίθενται ορισμένοι τίτλοι απαιτήσεων από το λογισμικό οι οποίες δεν ανήκουν στην ίδια εφαρμογή. Ταξινομήστε τις απαιτήσεις αυτές ως προς τις κατηγορίες απαιτήσεων από το λογισμικό που αναφέρθηκαν.**

1. Η εφαρμογή θα πρέπει να τρέχει σε περιβάλλον UNIX.
2. Η εφαρμογή θα πρέπει να υπολογίζει την κατανομή του πληθυσμού A στις κατηγορίες ΑΙ.Αη.
3. Η εφαρμογή θα πρέπει να μπορεί να χρησιμοποιείται ταυτόχρονα από περισσότερους του ενός χρήστες.
4. Η σχεδίαση θα πρέπει να καταγράφεται με χρήση του προτύπου UML.
5. Η εφαρμογή θα πρέπει να διαχειρίζεται αρχείο μαθητών και βαθμολογίας σε μαθήματα που αυτοί παρακολουθούν.
6. Η καταχώρηση της βαθμολογίας θα πρέπει να γίνεται μόνο από εξουσιοδοτημένους χρήστες.
7. Η εφαρμογή θα υλοποιηθεί σε γλώσσα C++ με τη βοήθεια του εργαλείου StP.
8. Η επικοινωνία της εφαρμογής με τη βάση δεδομένων θα γίνεται μέσω δικτύου TCP/IP.
9. Σε περίπτωση μη διαθεσιμότητας του δικτύου, η προσπάθεια σύνδεσης θα επαναλαμβάνεται ανά δύο λεπτά και επί μία ώρα.
10. Η εφαρμογή θα εκτυπώνει λίστα με τους εγγεγραμμένους μαθητές ανά μάθημα.

## ΕΝΟΤΗΤΑ 4.2. ΜΗΧΑΝΙΚΗ ΑΠΑΙΤΗΣΕΩΝ

Στην ενότητα αυτή θα παρουσιαστεί η γενική διαδικασία ανάλυσης και προσδιορισμού απαιτήσεων από το λογισμικό η οποία αναφέρεται και ως «μηχανική απαιτήσεων» (requirements engineering).

### 4.2.1. Εισαγωγή

Το πλήθος και η πολυπλοκότητα που χαρακτηρίζει πολλές από τις απαιτήσεις από το λογισμικό, ο σαφής προσδιορισμός και η παρακολούθηση των συσχετίσεων μεταξύ αυτών καθώς και με απαιτήσεις από το σύστημα και τα διαφορετικά επίπεδα λεπτομέρειας στην περιγραφή τους είναι μερικά από τα σημαντικότερα προβλήματα που συναντά κανείς όταν καλείται να αντιμετωπίσει ένα πρόβλημα προσδιορισμού απαιτήσεων από το λογισμικό. Είναι ευνόητο ότι η επιτυχής αντιμετώπιση ενός τέτοιου προβλήματος δεν μπορεί να γίνει παρά μόνο με πειθαρχία, ακολουθώντας συγκεκριμένα βήματα και καταγράφοντας τα αποτελέσματα που παράγονται σε κάθε βήμα.

Υπάρχουν διάφορες προσεγγίσεις στο πρόβλημα αυτό, καθεμία εκ των οποίων προτείνει τα δικά της βήματα ή τις δικές τις λεπτομέρειες εκτέλεσης κάθε βήματος. Στην πράξη, δεν υπάρχει μια λύση που είναι καλύτερη από άλλες, και κάθε κατασκευαστής λογισμικού ακολουθεί τελικά μια δική του εκδοχή που περιέχει στοιχεία μίας ή και περισσότερων προσεγγίσεων. Ο τρόπος επίλυσης του προβλήματος προσδιορισμού των απαιτήσεων που προτείνει καθεμία από αυτές αναφέρεται ως «μηχανική απαιτήσεων» (requirements engineering). Θα προσεγγίσουμε το θέμα της μηχανικής απαιτήσεων ως μια γενική αλληλουχία ενεργειών που πρέπει να γίνονται κατά τον προσδιορισμό των απαιτήσεων από το λογισμικό. Κατά τις ενέργειες αυτές παράγονται ορισμένα προϊόντα με τη μορφή εγγράφων και διαγραμμάτων.

Η προσέγγιση που θα ακολουθηθεί στο παρόν βιβλίο αναφέρεται ως «δομημένη ανάλυση» (structured analysis) και έχει γνωρίσει σημαντική διάδοση στο παρελθόν, όντας ένας αποτελεσματικός και πειθαρχημένος τρόπος για την κατασκευή λογισμικού. Τα τελευταία χρόνια έχει κερδίσει σημαντικό έδαφος η αντικειμενοστρεφής ανάλυση και σχεδίαση (object-oriented

analysis and design), η οποία μπορεί να θεωρηθεί ως υπερσύνολο της δομημένης προσέγγισης ανάπτυξης λογισμικού.

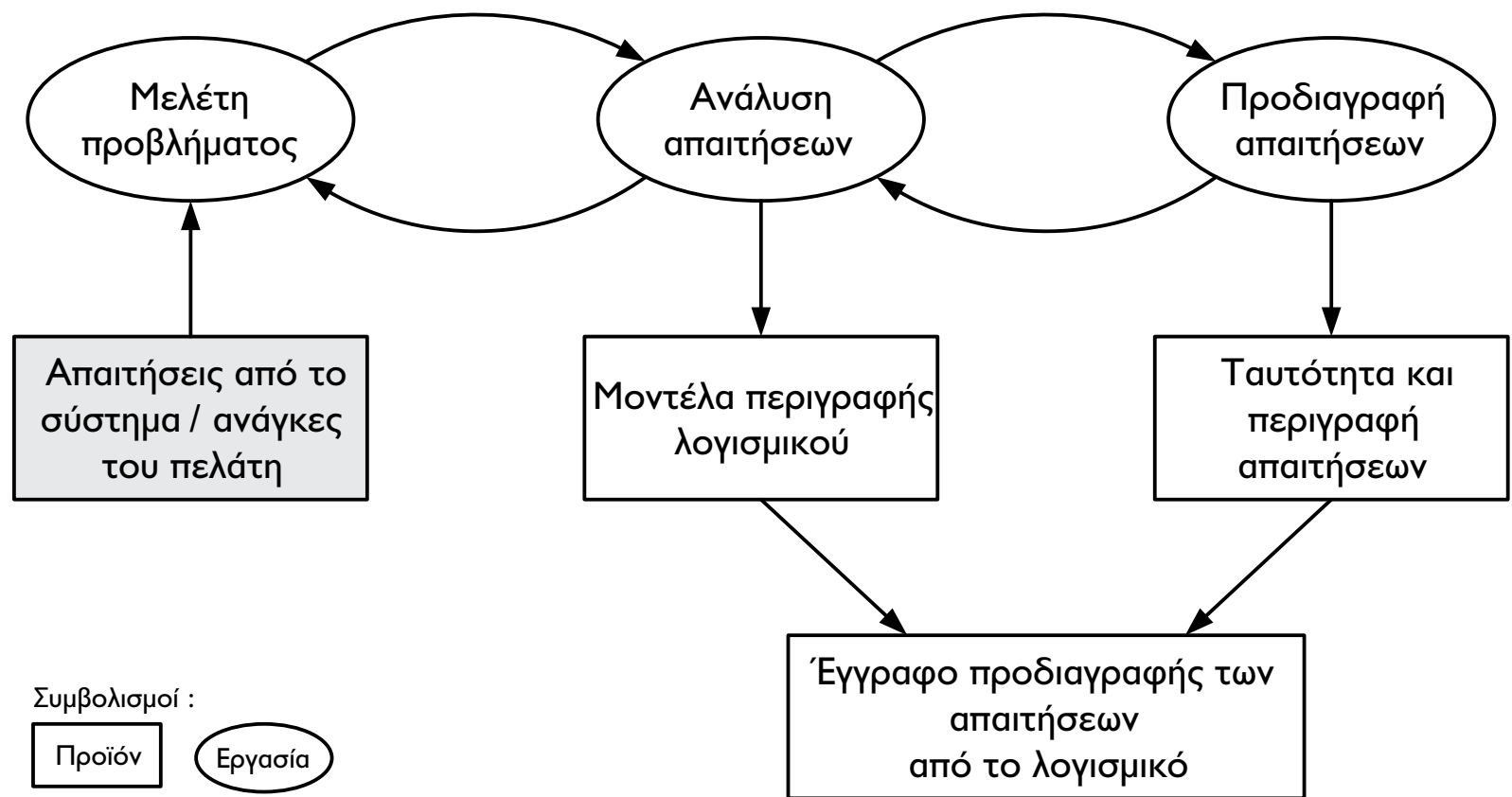
#### **4.2.2. Βήματα στον προσδιορισμό απαιτήσεων**

Η πρώτη πηγή για τον καθορισμό των απαιτήσεων από μια εφαρμογή λογισμικού είναι ασφαλώς ο πελάτης, ο οποίος περιγράφει στον κατασκευαστή τις εργασίες που θεωρεί απαραίτητο να εκτελούνται από το λογισμικό. Η περιγραφή αυτή συνήθως γίνεται με τη μορφή μιας έκθεσης, η οποία ενίοτε δεν είναι πλήρης και περιέχει ασάφειες και διφορούμενα. Η έκθεση αυτή αποτελεί το πρώτο υλικό που έχει στη διάθεσή του ο κατασκευαστής, προκειμένου να καθορίσει όλα τα στοιχεία που θα του επιτρέψουν να κατασκευάσει λογισμικό που ικανοποιεί τον πελάτη, καθώς και να καθορίσει το κόστος και να εκτιμήσει τον χρόνο που θα απαιτηθεί. Εκτός από τις εργασίες που θα πρέπει να εκτελεί το λογισμικό, πρέπει να προσδιοριστούν και άλλα χαρακτηριστικά του όπως, για παράδειγμα, το περιβάλλον λειτουργίας, ο τρόπος χρήσης και οι επιδόσεις.

Από την άλλη πλευρά, ο τρόπος με τον οποίο αντιλαμβάνεται ο πελάτης τις εργασίες που εκτελεί το λογισμικό δεν βρίσκεται πάντα σε αντιστοιχία με τον τρόπο με τον οποίο αυτές μπορούν να ενσωματωθούν σε μια εφαρμογή λογισμικού. Σε πολλές περιπτώσεις, αυτό που ο πελάτης αντιλαμβάνεται ως μία και μοναδική λειτουργία απαιτείται να αναλυθεί σε περισσότερες, προκειμένου να υλοποιηθεί στο λογισμικό. Από τα παραπάνω συνάγεται ότι μια πρόσφορη διαδικασία για τον καθορισμό των απαιτήσεων από το λογισμικό περιγράφεται ως μια ακολουθία βημάτων, σε καθένα από τα οποία παράγεται μία και ολοένα λεπτομερέστερη εκδοχή των απαιτήσεων από το λογισμικό. Τελικό προϊόν της διαδικασίας αυτής είναι το έγγραφο «προδιαγραφές των απαιτήσεων από το λογισμικό», καθώς και ένα σύνολο από διαγράμματα τα οποία το συνοδεύουν. Η γενική μορφή της διαδικασίας φαίνεται στο Σχήμα 4.3.



**Σχήμα 4.3** Μηχανική απαιτήσεων: Η γενική μορφή της διαδικασίας προσδιορισμού των απαιτήσεων από το λογισμικό.





Η διαδικασία τροφοδοτείται με το έγγραφο των απαιτήσεων από το σύστημα ή, αν αυτό δεν είναι διαθέσιμο, με μια έκθεση αναγκών του πελάτη. Το πρώτο βήμα είναι η **μελέτη του εγγράφου απαιτήσεων από το σύστημα ή/και των αναγκών του πελάτη**, η οποία στοχεύει στην αρχική κατανόηση του πεδίου του προβλήματος, στην επίλυση του οποίου καλείται να χρησιμοποιηθεί το λογισμικό που κατασκευάζεται. Η μελέτη αυτή συνήθως πραγματοποιείται από διοικητική και οργανωτική σκοπιά προκειμένου να εκτιμηθεί η βιωσιμότητα, τα ρίσκα, ο προϋπολογισμός, το χρονοδιάγραμμα και άλλες διαχειριστικές παράμετροι της ανάπτυξης λογισμικού.

Ακολουθεί η **ανάλυση των απαιτήσεων**, η οποία στοχεύει στη δημιουργία μοντέλων που περιγράφουν διαφορετικές πλευρές του λογισμικού. Τα μοντέλα αυτά παριστάνονται με τη βοήθεια διαγραμμάτων ροής δεδομένων, οντοτήτων – συσχετίσεων και μετάβασης καταστάσεων, καθώς και με χρήση ενός πίνακα λεξικού δεδομένων, αναλυτική αναφορά στα οποία θα γίνει στη συνέχεια του κεφαλαίου.

Η **διάκριση και προδιαγραφή κάθε συγκεκριμένης απαίτησης** από το λογισμικό είναι το επόμενο βήμα κατά το οποίο συμπληρώνεται το έγγραφο «προδιαγραφές των απαιτήσεων από το λογισμικό», το οποίο είναι το επιθυμητό αποτέλεσμα της διαδικασίας. Το έγγραφο αυτό περιγράφει με λεπτομέρεια τις απαιτήσεις από το λογισμικό, τις ταξινομεί και τις ιεραρχεί και βρίσκεται σε πλήρη συμφωνία με τα διαγράμματα που έχουν παραχθεί στο προηγούμενο βήμα. Μια προτεινόμενη δομή ενός τέτοιου εγγράφου θα παρουσιαστεί παρακάτω. Όπως φαίνεται στο Σχήμα 4.3, κατά τη διαδικασία αυτή μπορεί να πραγματοποιούνται πισωγυρίσματα όταν κάτι τέτοιο κρίνεται απαραίτητο.

### Άσκηση 3/Κεφάλαιο 4

Με βάση τα όσα έχουν αναφερθεί μέχρι το σημείο αυτό, περιγράψτε την ποιοτική διαφορά της εργασίας «ανάλυση απαιτήσεων» και της εργασίας «διάκριση και προδιαγραφή κάθε συγκεκριμένης απαίτησης».

## Μελέτη περίπτωσης/Κεφάλαιο 4

Ακολούθως θα παρουσιαστεί μια μελέτη περίπτωσης, δηλαδή μια εφαρμογή λογισμικού στην ανάπτυξη της οποίας θα αναφερόμαστε στη συνέχεια του κεφαλαίου, καθώς και σε επόμενα κεφάλαια του βιβλίου. Οι λειτουργίες και τα χαρακτηριστικά της εφαρμογής θα είναι σαφώς περιορισμένα σε σχέση με τις απαιτήσεις για χρήση σε πραγματικό χώρο, θα είναι όμως επαρκή για τους εκπαιδευτικούς σκοπούς της.

### **Εφαρμογή υποστήριξης εργασιών γραμματείας εκπαιδευτικής μονάδας.**

Περιγραφή προβλήματος από τον πελάτη.

Ο πελάτης μας είναι υπεύθυνος για τη λειτουργία της γραμματείας ενός υποθετικού εκπαιδευτικού φορέα. Στον φορέα του παραδείγματός μας, λόγω του πλήθους των σπουδαστών, των καθηγητών και των μαθημάτων, του όγκου και της πολυπλοκότητας των εργασιών υποστήριξης (αρχείου, εγγραφών κ.ά.), είναι αναγκαία η χρήση μιας εφαρμογής λογισμικού. Ο πελάτης αποφασίζει να ονομάσει την εφαρμογή αυτή «Επίκουρος» και να αναθέσει την ανάπτυξή της σε κατασκευαστή λογισμικού, τον ρόλο του οποίου παίζουμε εμείς.

Η εφαρμογή θα πρέπει να τηρεί αρχεία σπουδαστών, καθηγητών, μαθημάτων, εγγραφής σε μαθήματα, καθώς και αποτελέσματα βαθμολογίας. Η εφαρμογή θα πρέπει να εκτυπώνει καταστάσεις σπουδαστών, καθηγητών, μαθημάτων και βαθμολογίας με κριτήρια που θα δίνει ο χρήστης. Η εφαρμογή δεν θα πρέπει να επιτρέπει τη διαγραφή ενός σπουδαστή ή καθηγητή από το αρχείο αν αυτός έχει εγγραφεί ή διδάξει μάθημα αντίστοιχα. Το περιβάλλον λειτουργίας θα είναι ένας αυτόνομος ηλεκτρονικός υπολογιστής με Windows 98.

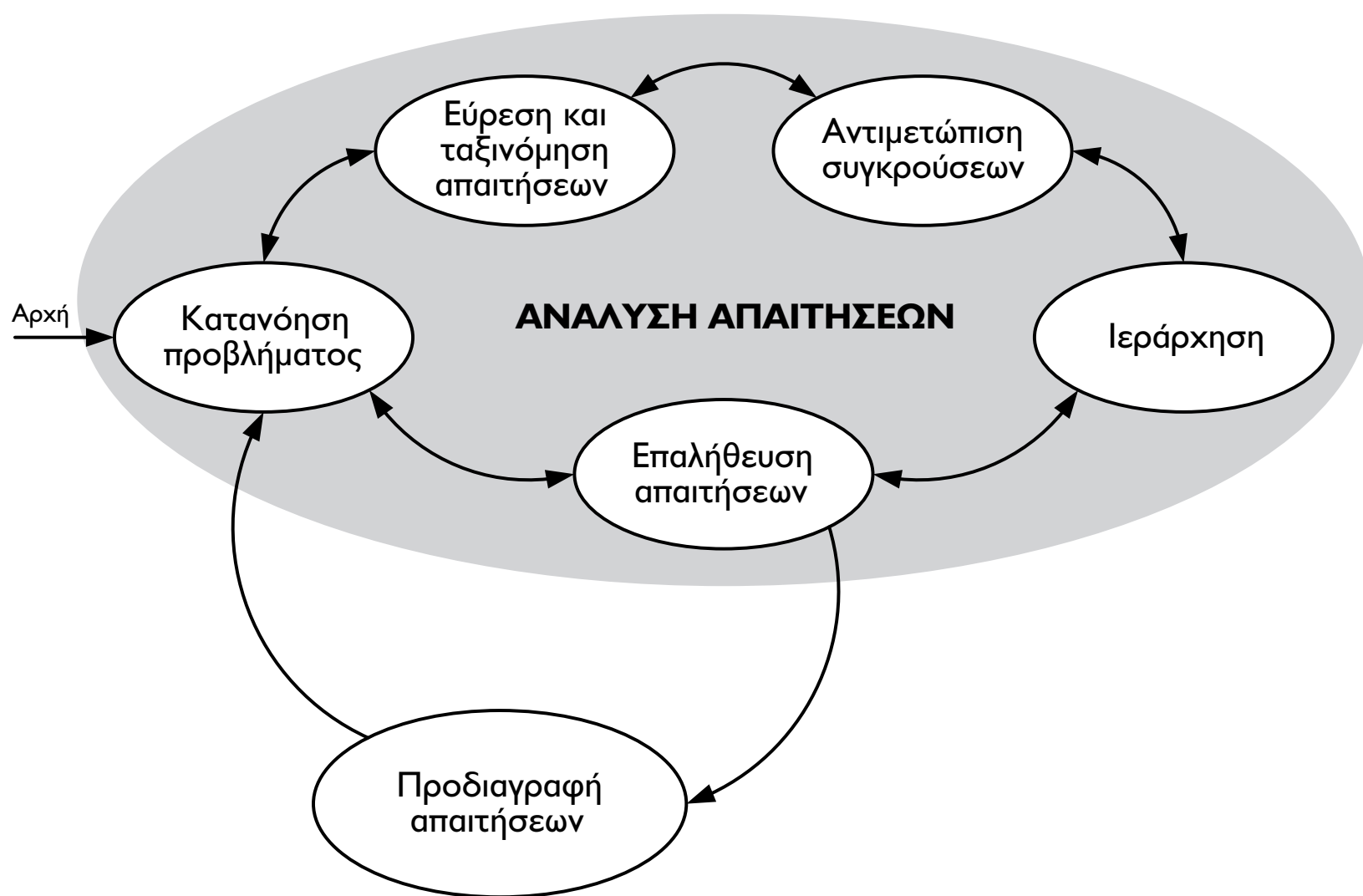
## ΕΝΟΤΗΤΑ 4.3. ΑΝΑΛΥΣΗ ΚΑΙ ΠΡΟΔΙΑΓΡΑΦΗ ΑΠΑΙΤΗΣΕΩΝ

Ακολούθως θα εξειδικευτεί η γενική διαδικασία που παρουσιάστηκε στην προηγούμενη ενότητα εστιάζοντας μέσα στις εργασίες ανάλυσης και προδιαγραφής των απαιτήσεων που φαίνονται στο Σχήμα 4.3 και περιγράφοντας τον τρόπο εκτέλεσής τους.

### 4.3.1. Ανάλυση απαιτήσεων

Κατά την ανάλυση των απαιτήσεων εντοπίζονται για πρώτη φορά οι απαιτήσεις από το λογισμικό και ακολουθούν έναν κύκλο ταξινόμησης, ιεράρχησης και επαλήθευσης, όπως φαίνεται στο Σχήμα 4.4. Αποτέλεσμα των εργασιών που εκτελούνται στη φάση αυτή είναι ένα σύνολο απαιτήσεων από το λογισμικό οι οποίες περιγράφονται με μορφή διαγραμμάτων. Η περιγραφή αυτή αποτελεί την είσοδο στο επόμενο βήμα, αυτό της διάκρισης και προδιαγραφής των απαιτήσεων από το λογισμικό.

**Σχήμα 4.4** Επιμέρους βήματα που εκτελούνται κατά τη φάση της ανάλυσης απαιτήσεων από το λογισμικό.



Κατά την **κατανόηση του προβλήματος**, ο αναλυτής υπεισέρχεται ο ίδιος στην ουσία του προβλήματος όσο περισσότερο γίνεται. Η έκθεση των αναγκών του πελάτη δεν είναι συνήθως επαρκής για την αντίληψη της ουσίας του αντικειμένου σχετικά με το οποίο αναπτύσσεται το λογισμικό. Χωρίς την επίτευξη από πλευράς του κατασκευαστή κάποιας εξοικείωσης με την ουσία του προβλήματος στην επίλυση του οποίου συμβάλλει το λογισμικό που κατασκευάζει, δεν είναι δυνατή η επιτυχής ανάπτυξη λογισμικού που αφορά την επίλυση αυτού. Η εξοικείωση αυτή θα πρέπει να γίνει στο αρχικό στάδιο ανάπτυξης και θα σημαδέψει ακολούθως τη διαδικασία ανάπτυξης του λογισμικού. Πρόκειται για μια συναρπαστική, ιδιαίτερα δημιουργική και, συνάμα, δύσκολη εργασία που καθιστά το επάγγελμα του μηχανικού λογισμικού από τα πιο ενδιαφέροντα και λιγότερο μονότονα.

Έπειτα **συλλέγονται οι απαιτήσεις** των εμπλεκομένων με το λογισμικό και γίνεται μια αρχική καταγραφή τους σε λίστα. Η συλλογή αυτή γίνεται με τη βοήθεια συνεντεύξεων, ερωτηματολογίων, συζητήσεων με ειδικούς ή με άλλους κατά περίπτωση πρόσφορους τρόπους. Η λίστα που δημιουργείται αρχικά περιέχει τις απαιτήσεις από το λογισμικό ατάκτως ερριμμένες. Στη συνέχεια, γίνεται μια πρώτη ταξινόμηση των απαιτήσεων σε ομάδες, ανάλογα με το υποσύνολο του προβλήματος που αφορούν ή με άλλο κατά περίπτωση πρόσφορο τρόπο.

Στο σημείο αυτό ενδεχομένως να εντοπίζονται ασυνέπειες, δηλαδή δύο ή περισσότερες απαιτήσεις των οποίων η ικανοποίηση δεν μπορεί να γίνει ταυτόχρονα, οπότε είναι αναγκαία η **επίλυση συγκρούσεων**, η οποία μπορεί να επαναφέρει στο προσκήνιο τις επαφές με τον πελάτη ή άλλη σχετική διαδικασία. Όταν έχει ολοκληρωθεί η επίλυση συγκρούσεων, οι απαιτήσεις τοποθετούνται σε μια **σειρά προτεραιότητας** ως προς τη σειρά ικανοποίησής τους. Η σειρά αυτή θα καθορίσει όχι μόνο τη χρονική αλληλουχία με την οποία ενσωματώνονται στο λογισμικό λειτουργίες που ικανοποιούν τις απαιτήσεις αλλά και το ποιες από αυτές δεν θα ικανοποιηθούν καθόλου, αν κάτι τέτοιο επιβληθεί από εξωτερικούς παράγοντες (λ.χ. κόστος).

Η διαδικασία ολοκληρώνεται με την **επαλήθευση των απαιτήσεων** όπως έχουν διαμορφωθεί και ιεραρχηθεί. Για να γίνει η επαλήθευση, συνήθως απαιτείται νέα επαφή με τον πελάτη με τη μορφή σύσκεψης ή ανταλλαγής

εγγράφων. Στην καλύτερη περίπτωση οι απαιτήσεις ικανοποιούν τον πελάτη και μπορεί να ξεκινήσει η κατασκευή των μοντέλων περιγραφής λογισμικού, που φαίνονται στο Σχήμα 4.3. Σε περίπτωση που οι απαιτήσεις δεν ικανοποιούν τον πελάτη, τότε λαμβάνουν χώρα τόσα πωγυρίσματα, όσα είναι αναγκαία, προκειμένου να γίνει αυτό. Δεν αποτελούν εξαίρεση οι περιπτώσεις που πρέπει κανείς να επανέλθει στη διαδικασία κατανόησης του προβλήματος και να ανακαλύψει νέες πλευρές ή/και νέες ερμηνείες.

#### **Μελέτη περίπτωσης/Κεφάλαιο 4**

Συνεχίζοντας τη μελέτη της εφαρμογής λογισμικού «Επίκουρος», παραθέτουμε μια αρχική καταγραφή απαιτήσεων από το λογισμικό. Η καταγραφή αυτή προέκυψε μετά από συνέντευξη με τον υποθετικό πελάτη.

1. Ο «Επίκουρος» θα τρέχει σε αυτόνομο υπολογιστή κάτω από το λειτουργικό σύστημα Windows 9x - 32 bit (95, 98, NT, 2000). Δεν απαιτείται σύνδεση σε δίκτυο.
2. Ζητείται η τήρηση αρχείων μαθητών, καθηγητών και μαθημάτων.
3. Κάθε μάθημα διδάσκεται από έναν καθηγητή σε κάθε ακαδημαϊκό έτος.
4. Κάθε σπουδαστής μπορεί να εγγράφεται σε κάθε μάθημα όσες φορές θέλει.
5. Κάθε σπουδαστής αξιολογείται σε μαθήματα στα οποία έχει εγγραφεί. Η αξιολόγηση αυτή μπορεί να γίνεται περισσότερες από μία φορές τόσο κατά τη διάρκεια του ακαδημαϊκού έτους (ενδιάμεση εξέταση), όσο και με τελικό γραπτό.
6. Δεν πρέπει να επιτρέπεται η καταχώρηση βαθμολογίας σε μάθημα στο οποίο δεν έχει γίνει εγγραφή.

7. Επιτρέπεται η διαγραφή σπουδαστή μόνο αν δεν έχει εγγραφεί σε κανένα μάθημα.
8. Επιτρέπεται η διαγραφή καθηγητή μόνο αν δεν έχει διδάξει κανένα μάθημα.
9. Επιτρέπεται η διαγραφή μαθήματος μόνο αν δεν έχουν υπάρξει εγγραφές ή εξετάσεις που να το αφορούν.
10. Ζητείται αλφαβητική εκτύπωση ολόκληρου του αρχείου των σπουδαστών.
11. Ζητείται αλφαβητική εκτύπωση των εγγεγραμμένων σε κάθε μάθημα σπουδαστών.
12. Ζητείται αλφαβητική εκτύπωση ολόκληρου του αρχείου καθηγητών.
13. Ζητείται αλφαβητική εκτύπωση της βαθμολογίας σε κάθε μάθημα.
14. Ζητείται η εκτύπωση της βαθμολογίας όλων των μαθημάτων για κάποιο συγκεκριμένο σπουδαστή.

#### 4.3.2. Προδιαγραφή απαιτήσεων

Όταν η παραπάνω διαδικασία ολοκληρωθεί, θα έχουν κατασκευαστεί οι πρώτες εκδοχές των διαγραμμάτων ροής δεδομένων, οντοτήτων – συσχετίσεων και μετάβασης καταστάσεων. Είναι η στιγμή που μπορεί να πραγματοποιηθεί η αναλυτική προδιαγραφή των απαιτήσεων με τη σύνταξη του εγγράφου «προδιαγραφές των απαιτήσεων από το λογισμικό».



## Προδιαγραφή:

Με την έννοια «προδιαγραφή» αναφερόμαστε στη δομημένη και λεπτομερή περιγραφή των απαιτήσεων από το λογισμικό, η οποία γίνεται με τη μορφή γραπτού λόγου και, όπου απαιτείται, διαγραμμάτων.

Το έγγραφο προδιαγραφών των απαιτήσεων από το λογισμικό είναι αναμφίβολα το σημαντικότερο από τα έγγραφα τεκμηρίωσης του λογισμικού. Οι ελλείψεις και οι αστοχίες όσων αναφέρονται σε αυτό θα μεταφερθούν σε όλη την υπόλοιπη διαδικασία κατασκευής του λογισμικού και ασφαλώς στο τελικό προϊόν, γεγονός που μπορεί να έχει ως αποτέλεσμα αυτό να είναι άχρηστο. Έχουν προταθεί αρκετοί εναλλακτικοί τρόποι δόμησης του εγγράφου προδιαγραφών των απαιτήσεων από το λογισμικό. Είναι γενικά αποδεκτό ότι μερικά επιθυμητά χαρακτηριστικά του εγγράφου αυτού είναι τα ακόλουθα:

- Θα πρέπει να περιγράφει τη συμπεριφορά του λογισμικού προς το εξωτερικό του περιβάλλον (χρήστης, άλλες εφαρμογές λογισμικού) και όχι εσωτερικά του στοιχεία.
- Θα πρέπει να περιγράφει όλους τους περιορισμούς που αφορούν την ανάπτυξη του λογισμικού.
- Θα πρέπει να είναι εύκολο να αλλαχτεί.
- Θα πρέπει να είναι χρήσιμο στη συντήρηση του λογισμικού.
- Θα πρέπει να περιγράφει τη συμπεριφορά του λογισμικού σε ανεπιθύμητες καταστάσεις.

Στο Σχήμα 4.5 παρατίθεται μια δομή του εγγράφου προδιαγραφών των απαιτήσεων από το λογισμικό, βασισμένη σε διεθνές πρότυπο του IEEE (830-1993), το οποίο αναφέρεται στη βιβλιογραφία.



## **I. Εισαγωγή**

- I.1 Ταυτότητα του εγγράφου
- I.2 Σκοπός
- I.3 Εμβέλεια
- I.4 Ορισμοί, ακρωνύμια, συντομογραφίες
- I.5 Πηγές αναφορών
- I.6 Περίληψη

## **2. Γενική περιγραφή του λογισμικού**

- 2.1 Στίγμα
- 2.2 Προοπτική
- 2.3 Γενικές λειτουργίες του λογισμικού
- 2.4 Χαρακτηριστικά χρηστών
- 2.5 Περιορισμοί
- 2.6 Παραδοχές και εξαρτήσεις

## **3. Ειδικές απαιτήσεις**

- 3.1 Απαιτήσεις εξωτερικών διεπαφών
  - 3.1.1 Διεπαφές χρήστη
  - 3.1.2 Διεπαφές υλικού
  - 3.1.3 Διεπαφές λογισμικού
  - 3.1.4 Διεπαφές επικοινωνιών
- 3.2 Λειτουργικές απαιτήσεις
  - 3.2.1 Τρόπος λειτουργίας I
    - 3.2.1.1 Λειτουργική απαίτηση I.1  
Περιγραφή, είσοδοι, επεξεργασία, έξοδοι
    - 3.2.1.2 Λειτουργική απαίτηση I.2  
Περιγραφή, είσοδοι, επεξεργασία, έξοδοι
    - ...

...

- 3.2.N. Τρόπος λειτουργίας N
  - 3.2.N.1 Λειτουργική απαίτηση N.1  
Περιγραφή, είσοδοι, επεξεργασία, έξοδοι
  - 3.2.N.2 Λειτουργική απαίτηση N.2  
Περιγραφή, είσοδοι, επεξεργασία, έξοδοι
  - ...

- 3.3 Απαιτήσεις επιδόσεων
- 3.4 Περιορισμοί σχεδίασης
  - 3.4.1 Περιορισμοί από το υλικό
  - 3.4.2 Συμμόρφωση με πρότυπα
- 3.5 Χαρακτηριστικά του λογισμικού
  - 3.5.1 Αξιοπιστία
  - 3.5.2 Διαθεσιμότητα
  - 3.5.3 Ασφάλεια
  - 3.5.4 Χαρακτηριστικά συντήρησης
  - 3.5.5 Μεταφερσιμότητα
- 3.6 Άλλες απαιτήσεις

**Σχήμα 4.5** Μια προτεινόμενη από το IEEE δομή εγγράφου προδιαγραφών των απαιτήσεων από το λογισμικό.

Η παραπάνω περιγραφή αφορά τη δομή, δηλαδή τα κεφάλαια και τα περιεχόμενα καθενός εξ' αυτών, τα οποία θα πρέπει να έχει το έγγραφο προδιαγραφών των απαιτήσεων από το λογισμικό. Το μόνο σχετικά νέο στοιχείο του εγγράφου είναι η δόμηση του Κεφαλαίου 3 ανάλογα με τον τρόπο λειτουργίας. Σε μια απλή εφαρμογή λογισμικού υπάρχει μόνο ένας τρόπος λειτουργίας (mode). Δεν ισχύει το ίδιο για μεγάλες εφαρμογές με ιδιαίτερες απαιτήσεις, οι οποίες μπορεί να έχουν περισσότερους του ενός τρόπους λειτουργίας. Η γενική δομή του εγγράφου, η οποία οφείλει να καλύπτει τη γενική περίπτωση, πρέπει να το προβλέψει αυτό. Παράδειγμα αποτελεί μια εφαρμογή συλλογής και επεξεργασίας δεδομένων σε πραγματικό χρόνο. Η συμπεριφορά της εφαρμογής τη στιγμή της συλλογής των δεδομένων, όπου οι απαιτήσεις σε επιδόσεις μπορεί να είναι ιδιαίτερα υψηλές, είναι εντελώς διαφορετική από τη συμπεριφορά της τη στιγμή της στατιστικής επεξεργασίας ή της συντήρησης των δεδομένων αυτών, οπότε διακρίνουμε δύο τρόπους λειτουργίας.

Για να επανέλθουμε, μελετώντας τη δομή του εγγράφου διαπιστώνουμε ότι όλες οι απαιτήσεις από το λογισμικό μπορούν να ταξινομηθούν, ώστε να περιγραφούν σε αυτό. Η ταξινόμηση αυτή δεν είναι πάντα εύκολη υπόθεση και συχνά δημιουργούνται συγχύσεις και ερωτήματα του τύπου «Τι πρέπει να γραφτεί σε αυτή την παράγραφο;». Η απάντηση δεν είναι εύκολη και απαιτείται αρκετή τριβή με το αντικείμενο, μέχρις ότου να μπορεί κανείς να νιώθει εμπιστοσύνη στον τρόπο με τον οποίο αντιμετωπίζει το θέμα. Η εμπειρία δείχνει ότι σε επόμενο στάδιο της ανάπτυξης θα φανεί το πόσο επαρκές είναι το έγγραφο προδιαγραφών των απαιτήσεων από το λογισμικό και θα δοθεί η ευκαιρία, ενδεχομένως με κάποια αναθεώρηση, αυτό να βελτιωθεί. Η ανάγκη για τέτοιες αναθεωρήσεις συνεχώς θα μειώνεται όσο αποκτάται εμπειρία.

## Μελέτη περίπτωσης/Κεφάλαιο 4

Η εφαρμογή λογισμικού «Επίκουρος» έχει μόνο έναν τρόπο λειτουργίας. Ακολούθως φαίνεται η παράγραφος 3.2 του εγγράφου προδιαγραφών των απαιτήσεων από το λογισμικό για τις απαιτήσεις 2, 7 και 13 του παραδείγματος που προηγήθηκε.

### 3.2.1 Λειτουργική απαίτηση A1

**Τήρηση αρχείου σπουδαστών:** Η εφαρμογή εμφανίζει φόρμα μέσω της οποίας ο χρήστης ενημερώνει το αρχείο σπουδαστών. **Είσοδος:** Στοιχεία σπουδαστών από το πληκτρολόγιο. **Επεξεργασία:** Ετοιμασία και επαλήθευση της εγγραφής. **Έξοδοι:** Ενημερωμένο αρχείο σπουδαστών ή μήνυμα λάθους.

### 3.2.2 Λειτουργική απαίτηση A2

**Τήρηση αρχείου καθηγητών:** Η εφαρμογή εμφανίζει φόρμα μέσω της οποίας ο χρήστης ενημερώνει το αρχείο καθηγητών. **Είσοδος:** Στοιχεία καθηγητών από το πληκτρολόγιο. **Επεξεργασία:** Ετοιμασία και επαλήθευση της εγγραφής. **Έξοδοι:** Ενημερωμένο αρχείο καθηγητών ή μήνυμα λάθους.

### 3.2.3 Λειτουργική απαίτηση A3

**Τήρηση αρχείου μαθημάτων:** Η εφαρμογή εμφανίζει φόρμα μέσω της οποίας ο χρήστης ενημερώνει το αρχείο μαθημάτων. **Είσοδος:** Στοιχεία μαθημάτων από το πληκτρολόγιο. **Επεξεργασία:** Ετοιμασία και επαλήθευση της εγγραφής. **Έξοδοι:** Ενημερωμένο αρχείο μαθημάτων ή μήνυμα λάθους.

### 3.2.4 Λειτουργική απαίτηση A4

**Διαγραφή σπουδαστή:** Η εφαρμογή εμφανίζει φόρμα στην οποία ο χρήστης δίνει τα στοιχεία του σπουδαστή που επιθυμεί να διαγράψει. **Είσοδος:** Στοιχεία σπουδαστή προς διαγραφή. **Επεξεργασία:** Έλεγχος ύπαρξης σπουδαστή, έλεγχος εγγραφής του σε μάθημα, έλεγχος συμμετοχής του σε εξέταση. **Εξοδοι:** Αρχείο με διαγραφμένες εγγραφές σπουδαστή ή μήνυμα λάθους.

### 3.2.5 Λειτουργική απαίτηση A5

**Εκτύπωση βαθμολογίας σε μάθημα:** Η εφαρμογή εμφανίζει διάλογο ερώτησης του κωδικού του μαθήματος, δέχεται τον κωδικό από το πληκτρολόγιο και ετοιμάζει την εκτύπωση, την οποία στέλνει στον εκτυπωτή. **Είσοδος:** Κωδικός μαθήματος. **Επεξεργασία:** Έλεγχος ύπαρξης μαθήματος και ύπαρξης εγγραφών βαθμολογίας, ετοιμασία και μορφοποίηση της εκτύπωσης. **Εξοδοι:** Εκτύπωση στον εκτυπωτή του συστήματος ή μήνυμα λάθους.

Θα πρέπει να παρατηρήσουμε ότι σε καθεμία από τις καταγεγραμμένες στο προηγούμενο βήμα απαιτήσεις μπορεί να αντιστοιχούν περισσότερες από μία παράγραφοι του εγγράφου προδιαγραφών των απαιτήσεων από το λογισμικό. Στο παράδειγμά μας, η απαίτηση 2 αντιστοιχεί στις λειτουργικές απαιτήσεις A1, A2 και A3, η απαίτηση 7 αντιστοιχεί στην A4 και, τέλος, η απαίτηση I3 αντιστοιχεί στην A5.

### **Δραστηριότητα 3/Κεφάλαιο 4**

Αφού έχετε μελετήσει την προηγούμενη ενότητα, επανεξετάστε την απάντηση που δώσατε στην άσκηση αυτοαξιολόγησης 3. Μπορείτε να αναπτύξετε τη διατύπωση της προηγούμενης απάντησης με γνώμονα το σε ποιον χώρο και από ποιον εκτελείται κάθε εργασία;

### **Δραστηριότητα 4/Κεφάλαιο 4**

Με βάση τα όσα αναφέρθηκαν μέχρι τώρα, εντοπίστε τουλάχιστον δύο αλληλοσυγκρουόμενα «επιθυμητά χαρακτηριστικά» του εγγράφου προδιαγραφών των απαιτήσεων από το λογισμικό.

### **Δραστηριότητα 5/Κεφάλαιο 4**

Αναπτύξτε, σε αναλογία με το τμήμα της μελέτης περίπτωσης που μόλις αναπτύχθηκε, τις απαιτήσεις 6, 10 και 14 του «Επίκουρος».

## ΕΝΟΤΗΤΑ 4.4. ΚΑΤΑΓΡΑΦΗ ΤΩΝ ΑΠΑΙΤΗΣΕΩΝ ΑΠΟ ΤΟ ΛΟΓΙΣΜΙΚΟ

Προκειμένου να αποδοθούν με κατανοητό τρόπο οι απαιτήσεις από το λογισμικό, δεν αρκεί η περιγραφή με ελεύθερο κείμενο αλλά απαιτείται και η χρήση μοντέλων τα οποία περιγράφουν το λογισμικό από διαφορετικές οπτικές γωνίες. Με την αναφορά στα μοντέλα που χρησιμοποιούνται στη δομημένη ανάλυση ασχολείται η ενότητα αυτή.

### 4.4.1. Εισαγωγή

Μια εφαρμογή λογισμικού μπορεί να ιδωθεί από πολλές διαφορετικές πλευρές, ανάλογα με το σημείο που εστιάζεται το ενδιαφέρον του παρατηρητή. Για παράδειγμα, μπορεί κανείς να έχει άποψη για μια εφαρμογή είτε παρατηρώντας τον τρόπο με τον οποίο γίνεται η διαχείριση των δεδομένων σε αυτή, είτε παρατηρώντας την εσωτερική δομή των δεδομένων, είτε τη συμπεριφορά της εφαρμογής προς το χρήστη, είτε και άλλα χαρακτηριστικά. Η παρατήρηση από όλες τις δυνατές οπτικές γωνίες προσεγγίζει την πλήρη περιγραφή της εφαρμογής, χωρίς καμία επιμέρους παρατήρηση να είναι επαρκής από μόνη της για τον σκοπό αυτό.

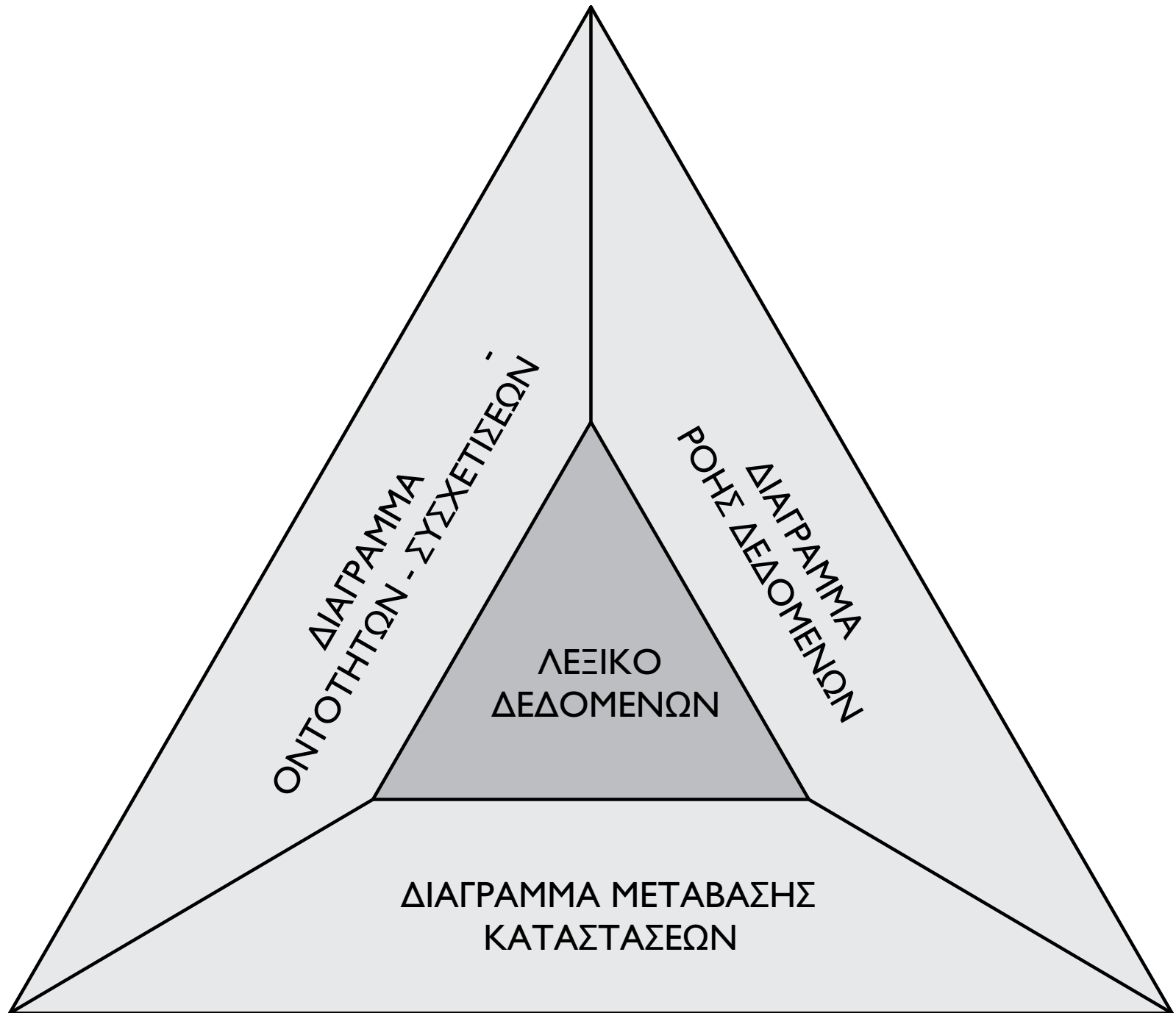
#### Μοντέλο παράστασης λογισμικού:

Ένα μοντέλο παράστασης λογισμικού είναι ένα διάγραμμα ή ένα σύνολο από ομοειδή διαγράμματα το οποίο περιγράφει το λογισμικό από μία συγκεκριμένη οπτική γωνία.

Η διατύπωση «συγκεκριμένη οπτική γωνία» στον προηγούμενο ορισμό δηλώνει ότι κανένα μοντέλο παράστασης λογισμικού δεν είναι πλήρες, δηλαδή δεν περιέχει όλες τις δυνατές πληροφορίες για το λογισμικό. Είναι μια αφαιρετική (abstract) περιγραφή κάποιων επιλεγμένων χαρακτηριστικών του και μόνο. Ο χαρακτηρισμός «αφαιρετική» έχει εδώ το νόημα ότι αφαιρούνται τα στοιχεία που δεν ενδιαφέρουν και περιγράφονται μόνο εκείνα στα οποία εστιάζεται η προσοχή.

Υπάρχουν πολλά μοντέλα παράστασης λογισμικού τα οποία προτείνονται από διαφορετικές μεθοδολογίες ανάπτυξης ή και από διαφορετικές εκδοχές της ίδιας μεθοδολογίας. Ακολουθώντας θα αναφερθούμε στα διαγράμματα ροής δεδομένων, οντοτήτων – συσχετίσεων, μετάβασης καταστάσεων, καθώς και στο λεξικό δεδομένων, όπως χρησιμοποιούνται στη δομημένη ανάλυση και σχεδίαση. Τα διαγράμματα αυτά περιγράφουν συμπληρωματικά μια εφαρμογή λογισμικού, όπως φαίνεται στο Σχήμα 4.6.

**Σχήμα 4.6** Συμπληρωματικότητα των μοντέλων παράστασης λογισμικού.





Η συμπληρωματικότητα έχει το νόημα ότι κανένα μοντέλο δεν περιγράφει πλήρως την εφαρμογή από μόνο του, ενώ όλα μαζί το κάνουν. Εκτός από συμπληρωματικά, τα μοντέλα που φαίνονται στο Σχήμα 4.6 πρέπει να είναι και *συνεπή* μεταξύ τους. Η *συνέπεια* έχει εδώ την έννοια ότι οι οντότητες που αναφέρονται σε κάθε τέτοιο μοντέλο δεν μπορεί να είναι εντελώς ξένες με αυτές που αναφέρονται στα υπόλοιπα. Η εξασφάλιση της απαίτησης αυτής επιτυγχάνεται με τη βοήθεια του *λεξικού δεδομένων*, το οποίο περιέχει αναφορές σε όλες τις οντότητες που περιλαμβάνονται στα μοντέλα παράστασης λογισμικού, όπως θα αναφερθεί αναλυτικότερα στη συνέχεια.

#### 4.4.2. Διαγράμματα ροής δεδομένων

Η έκθεση απαιτήσεων από το λογισμικό του πελάτη και, γενικά, μια πρώτη περιγραφή αυτών των απαιτήσεων χρησιμοποιεί φυσική γλώσσα και είναι, ασφαλώς, αρκετά χρήσιμη στον πελάτη ο οποίος θέτει τις απαιτήσεις. Δεν είναι, ωστόσο, το ίδιο χρήσιμη στον κατασκευαστή του λογισμικού ο οποίος χρειάζεται έναν πιο εποπτικό και, ταυτόχρονα, πιο κοντινό στην υλοποίηση του λογισμικού τρόπο. Ένας τέτοιος τρόπος είναι το διάγραμμα ροής δεδομένων το οποίο περιέχει τις απαιτήσεις του πελάτη με τη μορφή ενός δικτύου στο οποίο «ρέουν» δεδομένα τα οποία μετασχηματίζονται σε νέα δεδομένα από μονάδες λογισμικού. Κάθε μονάδα λογισμικού θεωρείται ως μετασχηματισμός που εφαρμόζεται επί κάποιων δεδομένων εισόδου, προκειμένου να δημιουργήσει νέα δεδομένα εξόδου. Τα διαγράμματα ροής δεδομένων χρησιμοποιούνται εκτεταμένα στη δομημένη ανάλυση και σχεδίαση και αποτελούν τη βάση για αρκετά από τα επόμενα βήματα της ανάπτυξης λογισμικού.

Ένα διάγραμμα ροής δεδομένων:

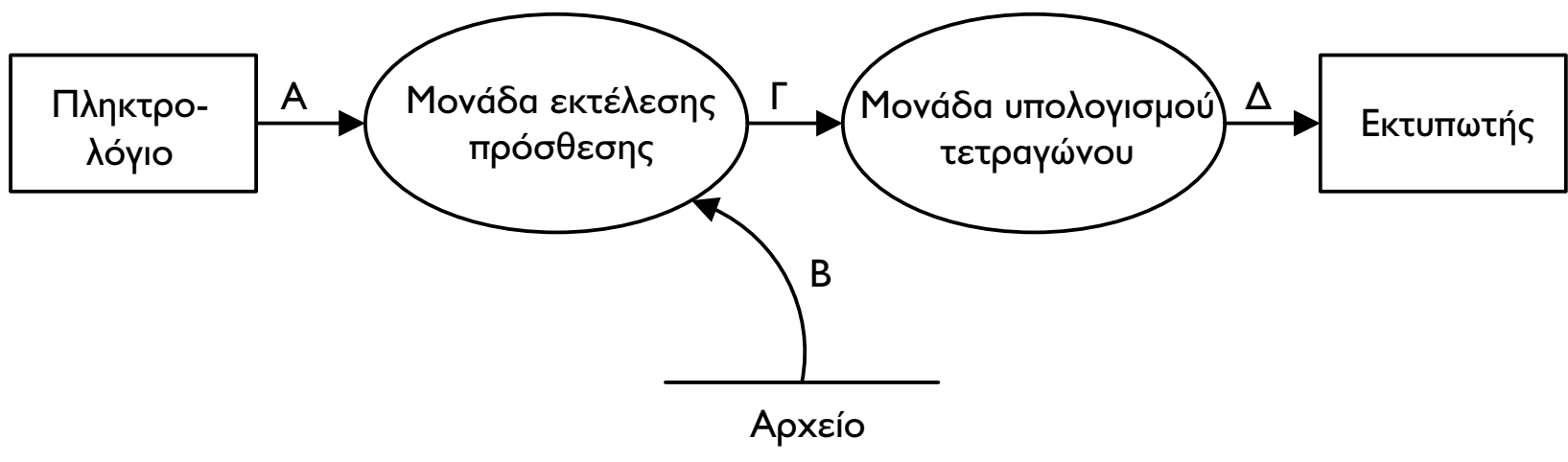
- Είναι εύκολα κατανοητό.
- Είναι ακριβές στο επίπεδο του *ποιες* λειτουργίες γίνονται και όχι στο *πώς*.
- Αποτελείται από πολλά διαφορετικά τμήματα τα οποία αφορούν επιμέρους τμήματα του λογισμικού.
- Μπορεί να σχεδιάζεται σε διαφορετικά επίπεδα λεπτομέρειας.

- Δεν περιέχει πληροφορία για τη χρονική αλληλουχία με την οποία συμβαίνουν οι μετασχηματισμοί δεδομένων.
- Είναι εύκολο να υποστεί μεταβολές, όταν κάτι τέτοιο κριθεί αναγκαίο.

### Παράδειγμα 3/Κεφάλαιο 4

Στο Σχήμα 4.7 φαίνεται ένα διάγραμμα ροής δεδομένων το οποίο περιγράφει την υλοποίηση της αριθμητικής πράξης  $(A+B)^2$ , όπου  $A$  και  $B$  πραγματικοί αριθμοί, εκ των οποίων ο  $A$  δίνεται από τον χρήστη και ο  $B$  διαβάζεται από κάποια αποθήκη δεδομένων (αρχείο).

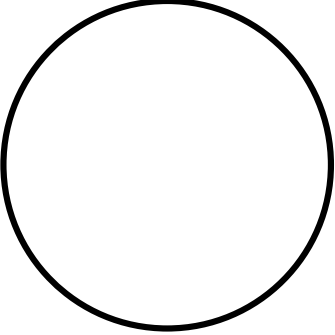



**Σχήμα 4.7** Ένα απλό διάγραμμα ροής δεδομένων.



Η «μονάδα εκτέλεσης πρόσθεσης» διαβάζει τους δύο αριθμούς (δεδομένα) και δημιουργεί έναν νέο ο οποίος αντιστοιχεί στο άθροισμά τους:  $\Gamma = A + B$ . Αυτό εισάγεται στη «μονάδα εκτέλεσης τετραγώνου» η οποία δημιουργεί το δεδομένο  $\Delta$  που αντιστοιχεί στην ύψωση του  $\Gamma$  στο τετράγωνο:  $\Delta = \Gamma^2 = (A + B)^2$ . Το δεδομένο αυτό στέλνεται στον εκτυπωτή.

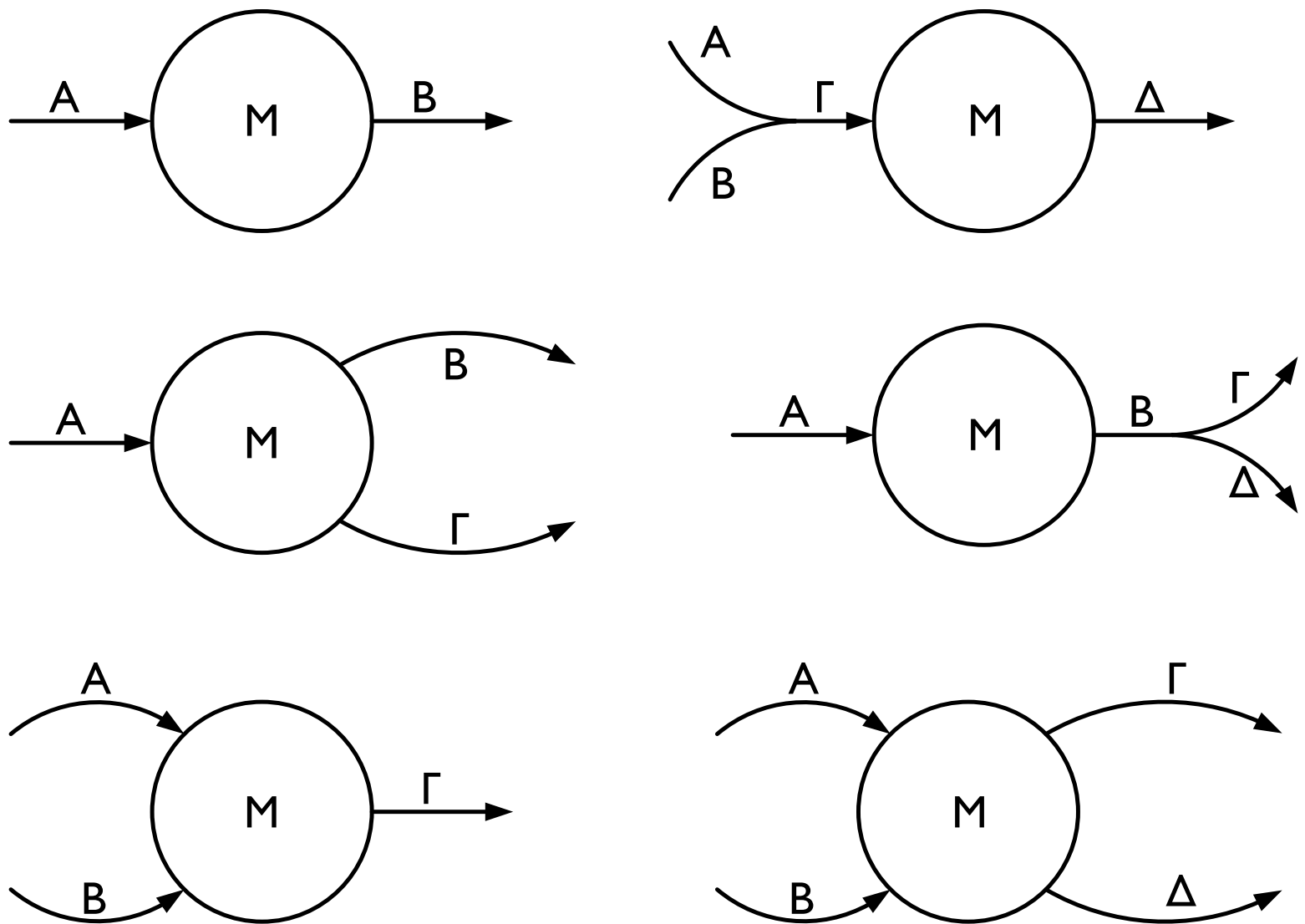
Οι συμβολισμοί που χρησιμοποιούνται στα διαγράμματα ροής δεδομένων φαίνονται στο σχήμα Σχήμα 4.8.

**Σχήμα 4.8** Συμβολισμοί διαγραμμάτων ροής δεδομένων.

Συμβολισμοί διαγραμμάτων ροής δεδομένων	
	Διαδικασία / μετασχηματισμός δεδομένων
	Εξωτερική πηγή ή αποδέκτης δεδομένων
	Ροή δεδομένων
	Αποθήκη δεδομένων

Κάθε υπολογιστική μονάδα–μετασχηματισμός δεδομένων παριστάνεται με έναν κύκλο (για ευκολία στη σχεδίαση χρησιμοποιείται και το ελλειψοειδές σχήμα). Κάθε παραγωγός ή αποδέκτης δεδομένων παριστάνεται με ένα παραλληλόγραμμο. Ως αποδέκτες ή παραγωγοί δεδομένων νοούνται οντότητες εξωτερικές προς το σύστημα λογισμικού, όπως ο άνθρωπος (χρήστης), ένας εκτυπωτής ή ένα ανεξάρτητο σύστημα λογισμικού. Μια ροή δεδομένων περιγράφεται με ένα βέλος που ενώνει μετασχηματισμούς μεταξύ τους ή με αποδέκτες/παραγωγούς δεδομένων. Πάνω από το βέλος σημειώνεται μια ονομασία για τα δεδομένα που αφορά η ροή. Οι συμβάσεις που ακολουθούνται στην καταγραφή των ροών δεδομένων φαίνονται στο Σχήμα 4.9.

**Σχήμα 4.9** Συμβάσεις παράστασης ροών σε διαγράμματα ροής δεδομένων.

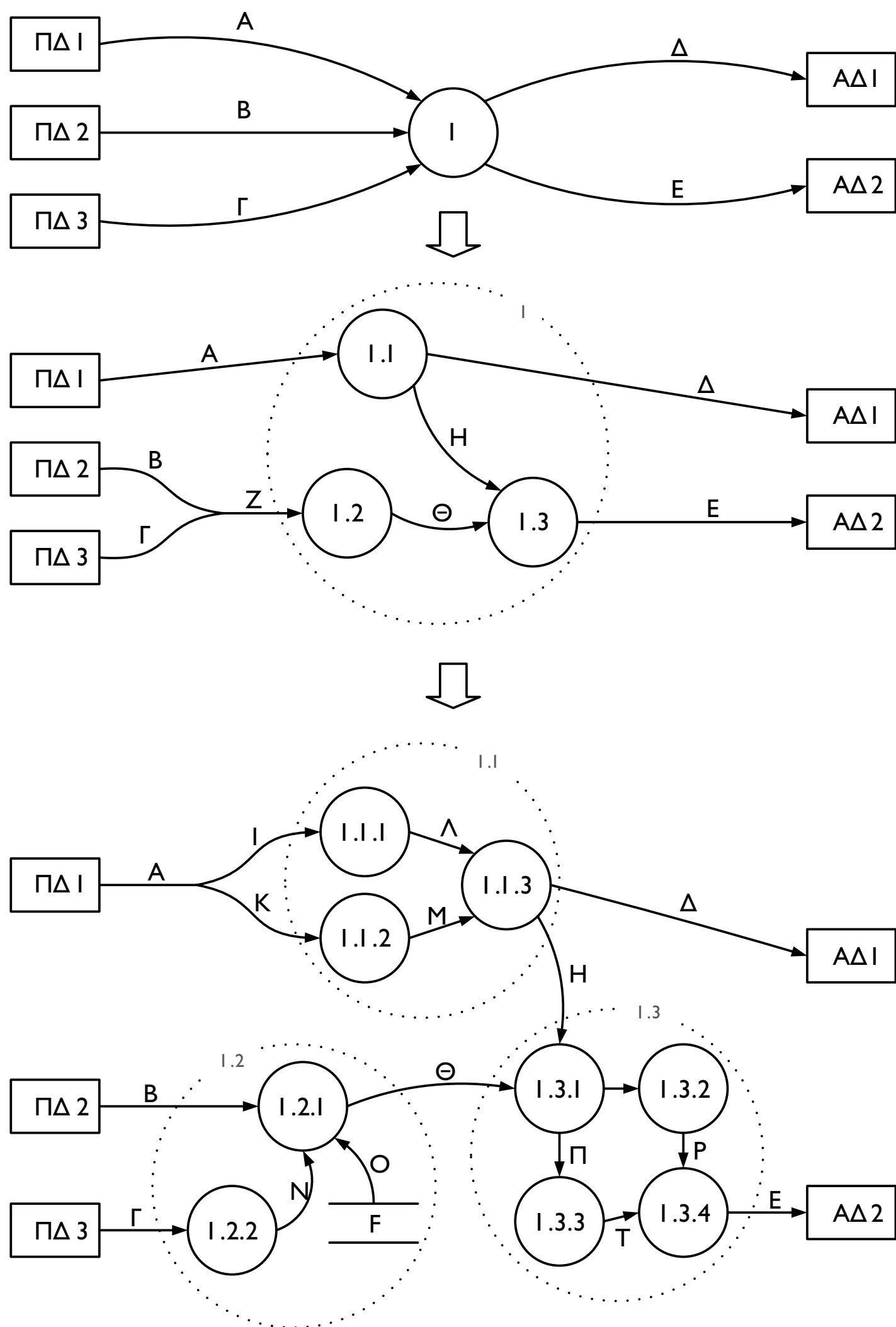


Στις συμβάσεις που φαίνονται στο προηγούμενο σχήμα θα πρέπει να σημειώσουμε τη δυνατότητα δημιουργίας σύνθετων από απλούστερα δεδομένα (πάνω δεξιά), καθώς και την ανάλυση σύνθετων σε απλούστερα (κέντρο δεξιά). Ένα απλό παράδειγμα είναι το ονοματεπώνυμο το οποίο μπορεί να συντίθεται από τα απλούστερα δεδομένα «όνομα» και «επώνυμο» ή να αναλύεται σε αυτά. Στη σύνθεση και αποσύνθεση δεδομένων δεν πρέπει να απαιτείται μετασχηματισμός, αλλά τα δεδομένα να είναι άμεσα διαθέσιμα ως ροές.

Το διάγραμμα ροής δεδομένων παριστάνεται σε διαφορετικά επίπεδα λεπτομέρειας. Στο πρώτο, λιγότερο λεπτομερές επίπεδο, ολόκληρη η εφαρμογή λογισμικού παριστάνεται ως ένας μετασχηματισμός σύνθετων δεδομένων. Ο μετασχηματισμός αυτός αναλύεται σε περισσότερα επίπεδα λεπτομέρειας, μέχρι το σημείο που ο κατασκευαστής κρίνει ικανοποιητικό. Κατά τη μετάβαση από ένα επίπεδο λεπτομέρειας σε ένα επόμενο (μεγαλύτερης λεπτομέρειας) πρέπει να παραμένουν συνεπείς οι εισερχόμενες και εξερχόμενες ροές δεδομένων. Ο αναγνώστης καλείται να διαπιστώσει τη συνέπεια αυτή στο παράδειγμα που φαίνεται στο Σχήμα 4.10.



**Σχήμα 4.10** Διάγραμμα ροής δεδομένων σε διαδοχικά επίπεδα λεπτομέρειας.



Μια πραγματική εφαρμογή λογισμικού μπορεί να περιέχει χιλιάδες ροές δεδομένων, των οποίων ακόμα και η σκέψη παράστασης με διάγραμμα προκαλεί δέος. Αυτό που ενδιαφέρει, ωστόσο, στα διαγράμματα ροής δεδομένων δεν είναι η απεικόνιση και της παραμικρής λεπτομέρειας αλλά των γενικών ποιοτικών στοιχείων των ροών δεδομένων και των μετασχηματισμών που εφαρμόζει σε αυτά μια συγκεκριμένη εφαρμογή λογισμικού. Μερικές απλές συμβουλές για την κατασκευή διαγραμμάτων ροής δεδομένων είναι οι ακόλουθες:

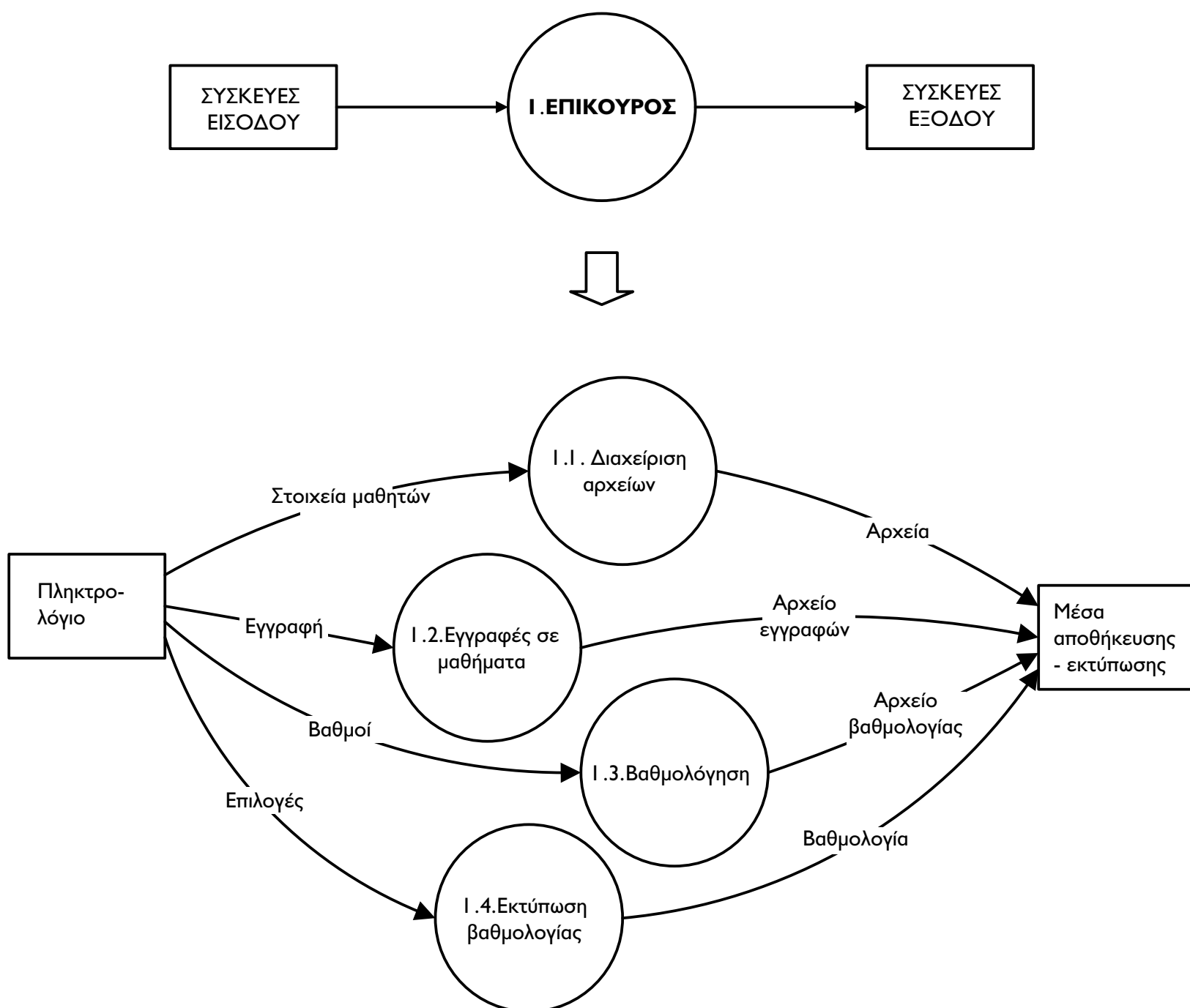
- Μην επικεντρώνετε την προσοχή σας στα αδιάφορα για την ουσία της εφαρμογής δεδομένα. Για παράδειγμα, το να συμπεριλάβει κανείς μια προσωρινή μεταβλητή μνήμης σε ένα διάγραμμα ροής δεδομένων δεν προσθέτει λεπτομέρεια αλλά μάλλον σύγχυση.
- Κατά τη μετάβαση από ένα επίπεδο λεπτομέρειας στο επόμενο, εκτός από τις διαδικασίες, είναι ενδεχόμενο να αποσυντίθενται και οι ροές δεδομένων. Η αποσύνθεση αυτή, αν και μπορεί να προκαλεί σύγχυση, είναι συνήθως χρήσιμη, ιδιαίτερα εκεί όπου η δημιουργία διαγραμμάτων έχει περιορισμούς χώρου. Σε κάθε περίπτωση, οι αποσυνθέσεις δεδομένων πρέπει να φαίνονται στο λεξικό δεδομένων.
- Μη συγχέετε τη ροή δεδομένων με τη διαγραμματική παράσταση αλγορίθμων ή δέντρων απόφασης και, γενικά, με οποιαδήποτε κατασκευαστική λεπτομέρεια που εξαρτάται από τη γλώσσα προγραμματισμού, το περιβάλλον λειτουργίας κ.ά. Αν υπάρχει ανάγκη, αυτά παριστάνονται σε επόμενο στάδιο της ανάπτυξης.
- Μην ανησυχείτε για την παράσταση της χρονικής αλληλουχίας με την οποία λαμβάνουν χώρα οι μετασχηματισμοί. Αυτή δεν παριστάνεται με διάγραμμα ροής δεδομένων αλλά με διάγραμμα μετάβασης καταστάσεων ή με κάποιου είδους ψευδοκώδικα, όπως θα δούμε στη συνέχεια.
- Μην επιχειρείτε να παραστήσετε τη ροή ελέγχου, δηλαδή την αλληλουχία εμφάνισης των οθονών, των μενού και, γενικά, της συμπεριφοράς του λογισμικού κατά την αλληλεπίδρασή του με το χρήστη. Αυτή παριστάνεται με διάγραμμα μετάβασης καταστάσεων, όπως θα φανεί στη συνέχεια.

- Διατηρείτε μια ισορροπία μεταξύ λεπτομέρειας και αφαίρεσης. Ένα ιδιαίτερα λεπτομερές διάγραμμα ροής δεδομένων είναι μεγάλο, διαβάζεται δύσκολα και επιδέχεται τροποποιήσεις ακόμα πιο δύσκολα. Από την άλλη, ένα απλό τέτοιο διάγραμμα τροποποιείται εύκολα, όμως δεν είναι ιδιαίτερα χρήσιμο.
- Προτιμάτε να κατασκευάζετε διαγράμματα ροής δεδομένων χρησιμοποιώντας κάποιο εργαλείο κατασκευασμένο για το σκοπό αυτό (εφόσον έχετε κάτι τέτοιο στη διάθεσή σας) και όχι απλά σχεδιαστικά προγράμματα. Στις περισσότερες περιπτώσεις, ο χρόνος εκμάθησης τέτοιων εργαλείων ανταποδίδεται σε παραγωγικότητα και ευελιξία.

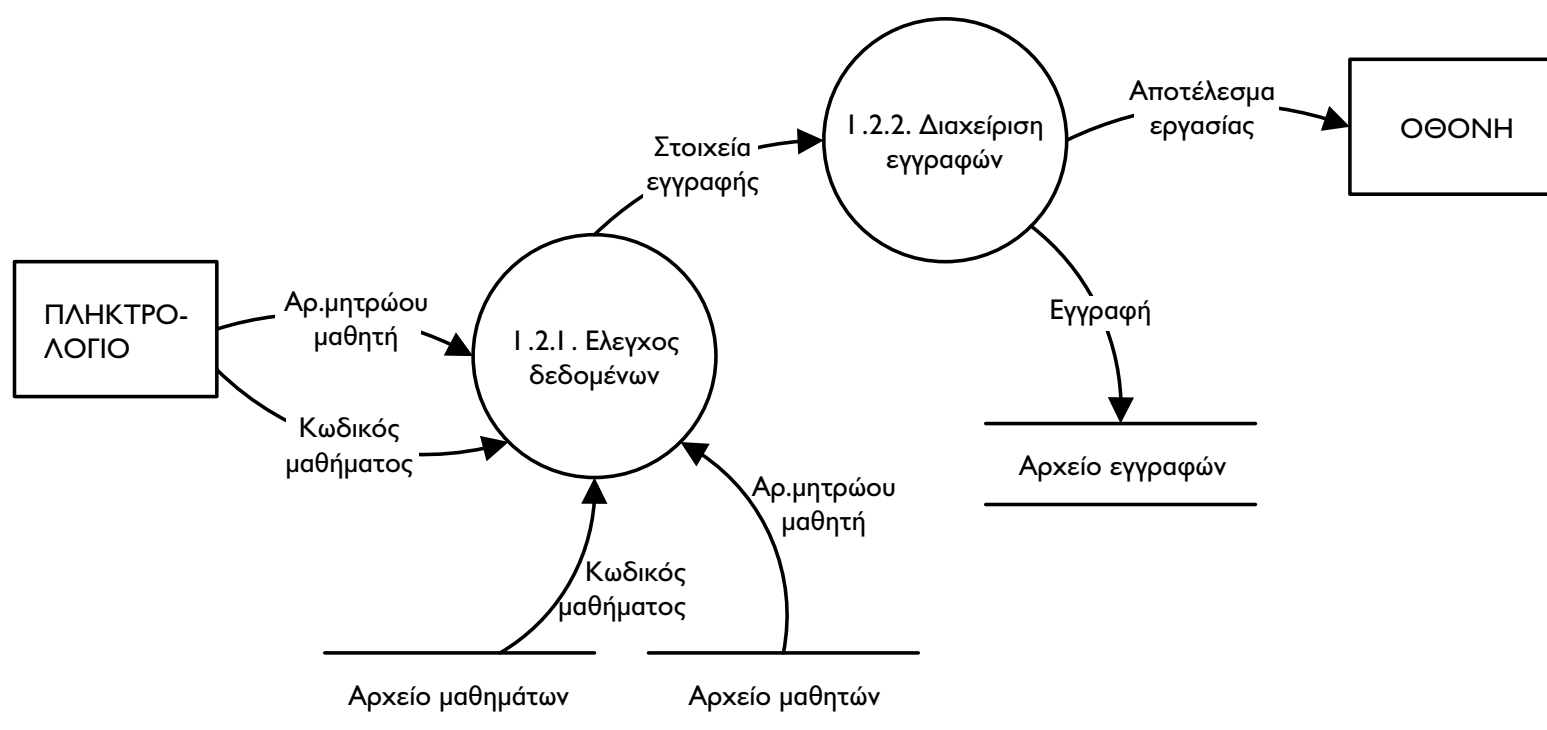
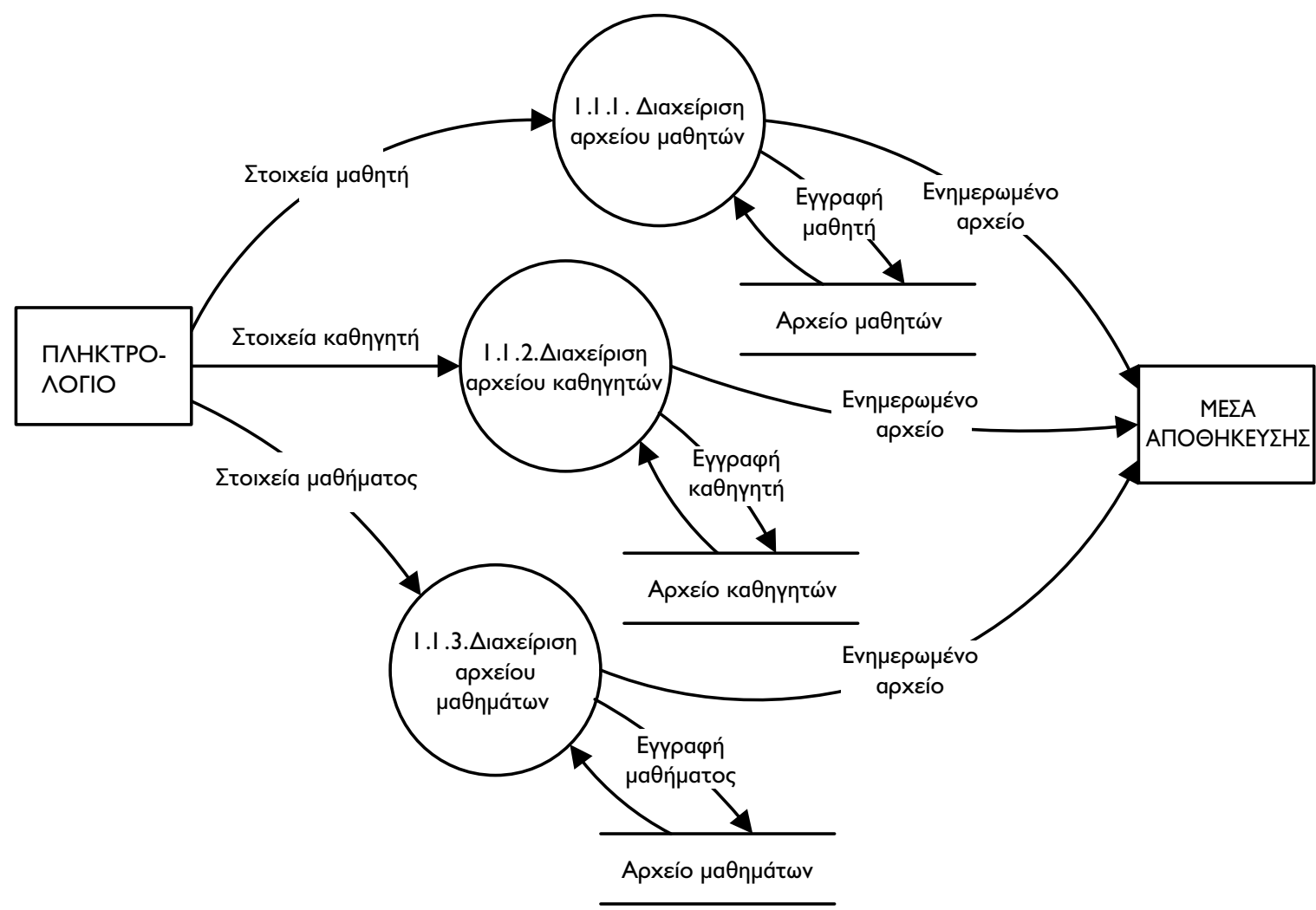
#### **Μελέτη περίπτωσης/Κεφάλαιο 4**

Με βάση τα όσα αναφέρονται στα Σχήματα 4.11 έως 4.13, παραθέτουμε τα διαγράμματα ροής δεδομένων της εφαρμογής λογισμικού «Επίκουρος», η οποία εισήχθη στην παράγραφο 4.2.2.

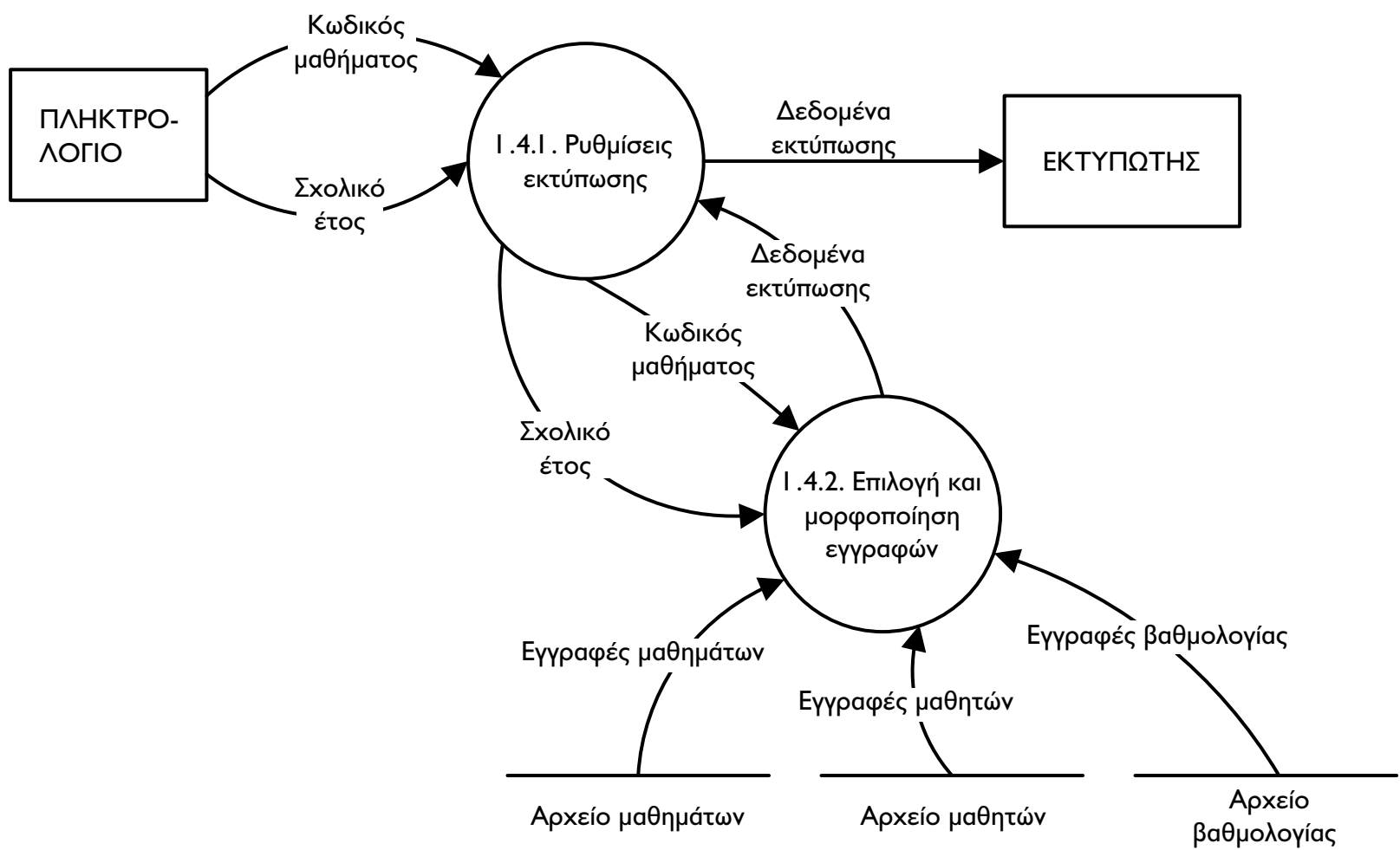
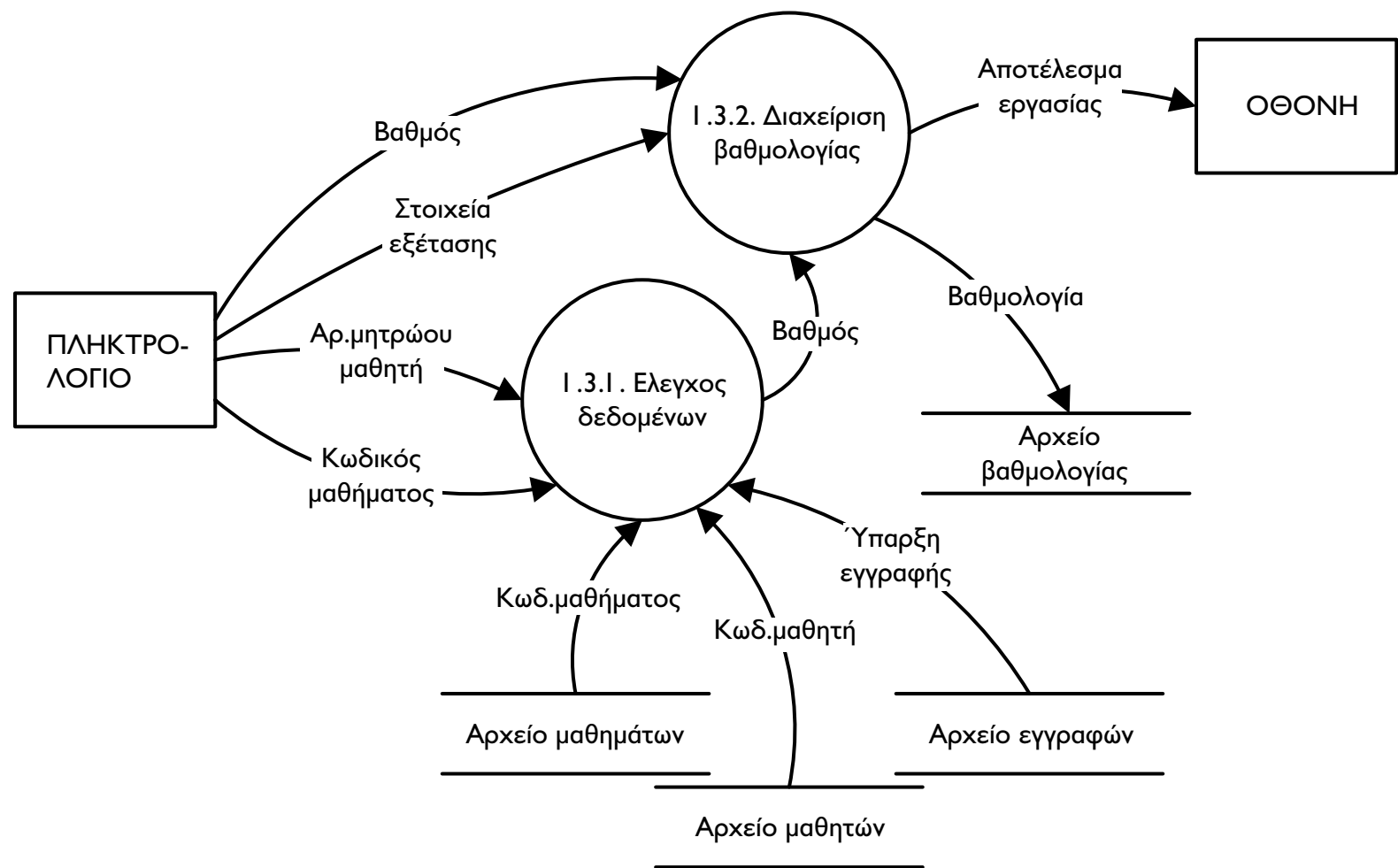
**Σχήμα 4.11** Τα διαγράμματα ροής δεδομένων της εφαρμογής λογισμικού «Επίκουρος» στο πρώτο και δεύτερο επίπεδο.



**Σχήμα 4.12** Τα διαγράμματα ροής δεδομένων της εφαρμογής λογισμικού «Επίκουρος» στο πρώτο και δεύτερο επίπεδο.



**Σχήμα 4.13** Επίπεδο 3 των διαγραμμάτων ροής δεδομένων της εφαρμογής λογισμικού «Επίκουρος» (μέρος 2).



#### **Δραστηριότητα 6/Κεφάλαιο 4**

Ο αναγνώστης καλείται να αντιστοιχίσει τους μετασχηματισμούς που εμφανίζονται στο Σχήμα 4.11, στο Σχήμα 4.12, καθώς και στο Σχήμα 4.13 με την πρώτη καταγραφή των απαιτήσεων σε λίστα (παράγραφος 4.3.1).

#### **Δραστηριότητα 7/Κεφάλαιο 4**

Εντοπίστε την αποσύνθεση δεδομένων στο Σχήμα 4.12 και στο Σχήμα 4.13 κατά τη μετάβαση από το δεύτερο επίπεδο λεπτομέρειας (Σχήμα 4.11) στο τρίτο.

#### **Δραστηριότητα 8/Κεφάλαιο 4**

Κατασκευάστε το διάγραμμα ροής δεδομένων για τη λειτουργική απαίτηση A5 (διαγραφή σπουδαστή) του συστήματος «Επίκουρος» (παράγραφος 4.3.2).

### **4.4.3. Διαγράμματα οντοτήτων – συσχετίσεων**

Για κάθε πρακτικό σκοπό μπορούμε να δεχτούμε ότι κάθε εφαρμογή λογισμικού σχετίζεται με τη διαχείριση κάποιων μόνιμα αποθηκευμένων δεδομένων. Η σχεδίαση και διαχείριση μόνιμα αποθηκευμένων δεδομένων αποτελεί αυτοτελές γνωστικό πεδίο από μόνη της και δεν θα μας απασχολήσει σε βάθος. Επειδή όμως η διαχείριση κάποιων μόνιμα αποθηκευμένων δεδομένων αποτελεί απαίτηση από μια εφαρμογή λογισμικού, θα αναφερθούμε στην παράσταση των δεδομένων και των συσχετίσεων μεταξύ αυτών με τη βοήθεια του διαγράμματος οντοτήτων – συσχετίσεων (entity-relationship diagram). Το διάγραμμα οντοτήτων – συσχετίσεων περιγράφει τις οντότητες δεδομένων και τις συσχετίσεις μεταξύ αυτών, σύμφωνα με το σχεσιακό μοντέλο δεδομένων.

Σημείωση: Ακολουθώντας, ο όρος «συσχέτιση» θα χρησιμοποιείται ισοδύναμα με τον όρο «σχέση». Στην ελληνική βιβλιογραφία ο όρος «σχέση» χρησιμοποιείται και για να δηλώσει την έννοια του πίνακα (table), γεγονός που δημιουργεί σύγχυση. Σε κάθε περίπτωση, ο αναγνώστης παραπέμπεται σε αυτοτελή μελέτη του γνωστικού αντικειμένου «Βάσεις Δεδομένων».

Ως οντότητα (entity) νοείται ένα σύνολο από αντικείμενα, πρόσωπα ή γεγονότα του πραγματικού κόσμου τα οποία βρίσκονται εντός του πεδίου ενδιαφέροντος της εφαρμογής λογισμικού η οποία κατασκευάζεται. Κάθε οντότητα χαρακτηρίζεται από ένα σύνολο στοιχείων, τα οποία ονομάζονται πεδία (fields), και περιέχουν τις τιμές ορισμένων χαρακτηριστικών ιδιωμάτων της οντότητας. Κάθε πεδίο περιέχει μια συγκεκριμένη πληροφορία που αφορά μια οντότητα. Το σύνολο των πεδίων που αφορούν μια συγκεκριμένη οντότητα ονομάζεται εγγραφή (record). Το σύνολο των εγγραφών αποθηκεύεται με τη βοήθεια ενός πίνακα (table). Για παράδειγμα, η οντότητα «καθηγητής» μπορεί να χαρακτηρίζεται από τα πεδία «αριθμός ταυτότητας», «όνομα», «επώνυμο», «διεύθυνση» και «τηλέφωνο», όπως φαίνεται στο Σχήμα 4.14.



Σχήμα 4.14 Πίνακας με πεδία και εγγραφές.

ΚΑΘΗΓΗΤΗΣ ← Οντότητα

ΑΡ.ΤΑΥΤ	ΟΝΟΜΑ	ΕΠΩΝΥΜΟ	ΔΙΕΥΘΥΝΣΗ	ΤΗΛ
ΑΙ23456	Βασίλειος	Βασιλείου	Λέσβου Ι	5554432
Α65432Ι	Αντώνης	Αντωνίου	Νίκης 22	9876543
ΜΙ95828	Γεώργιος	Γεωργίου	Βουλής 2Ι	Ι234567
...	...	...	...	...

← Ονόματα πεδίων

← Εγγραφή

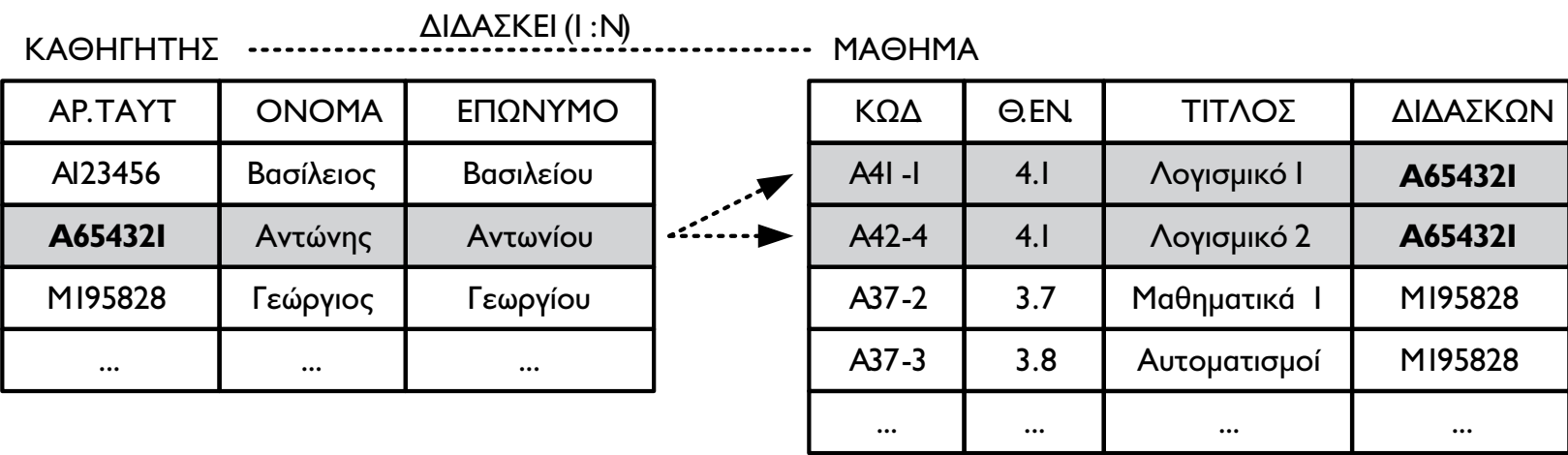
Πεδίο εγγραφής

Ως *συσχέτιση* (relationship) νοείται μια αντιστοίχιση μεταξύ διαφορετικών οντοτήτων η οποία περιγράφεται με ένα ρήμα. Στο παράδειγμα της εφαρμογής λογισμικού «Επίκουρος», μεταξύ των οντοτήτων «καθηγητής» και «μάθημα» μπορεί να αναγνωριστεί η συσχέτιση «διδάσκει». Επιπλέον της ύπαρξης της συσχέτισης, μας ενδιαφέρει και ο ποσοτικός της χαρακτήρας, δηλαδή το πλήθος των μελών (εγγραφών) της μιας οντότητας που μπορούν να συσχετίζονται με μέλη της άλλης. Στο σχεσιακό μοντέλο δεδομένων υπάρχουν τρία είδη συσχετίσεων:

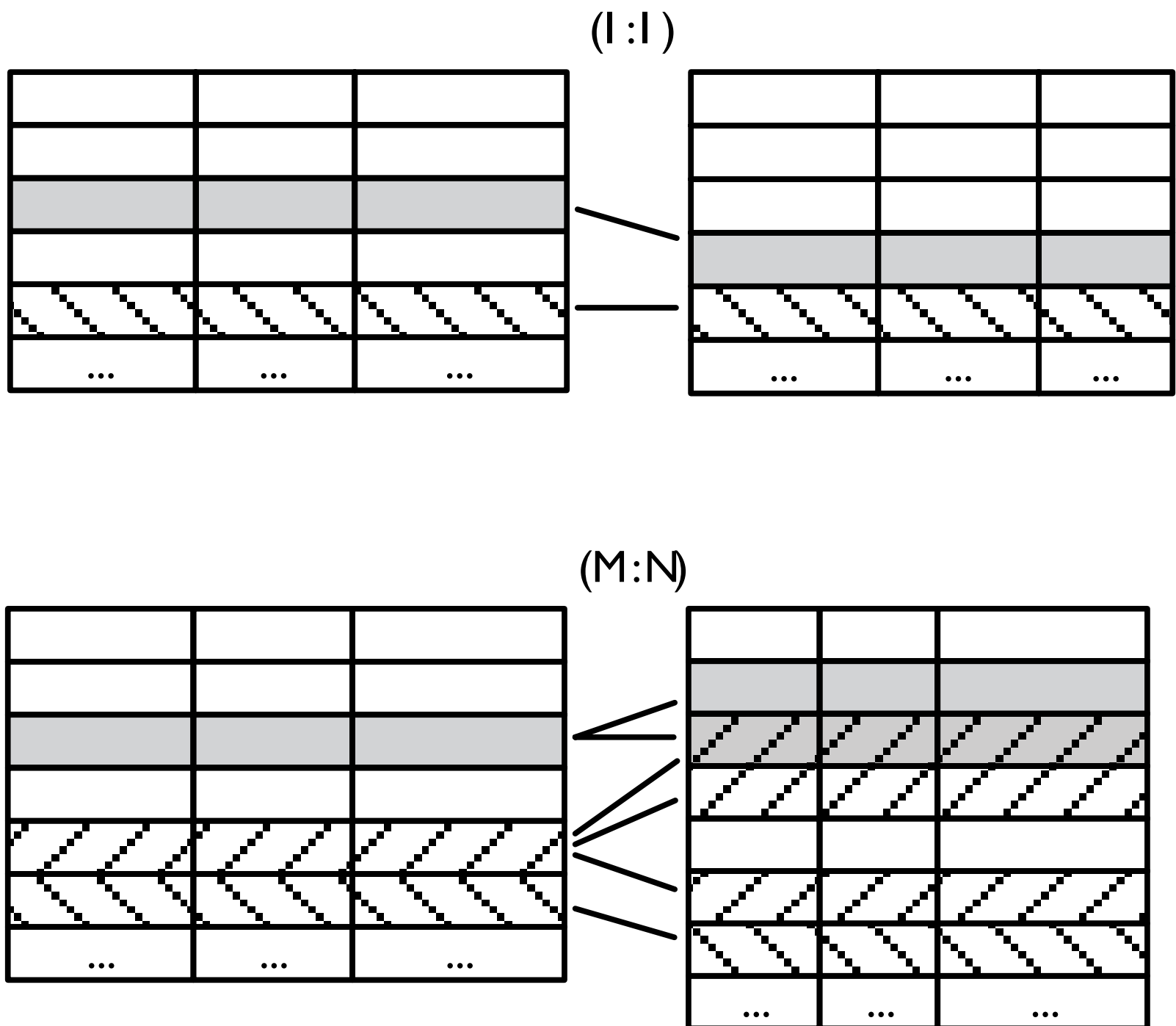
- Η συσχέτιση ένα-προς-ένα (1:1), στην οποία ένα μέλος του πληθυσμού μιας οντότητας A συσχετίζεται με/αντιστοιχεί σε ακριβώς ένα μέλος του πληθυσμού μιας οντότητας B.
- Η συσχέτιση ένα-προς-πολλά (1:N), στην οποία ένα μέλος του πληθυσμού μιας οντότητας A συσχετίζεται με/αντιστοιχεί σε τουλάχιστον ένα μέλος του πληθυσμού μιας οντότητας B.
- Η συσχέτιση πολλά-προς-πολλά (M:N), στην οποία ένα ή περισσότερα μέλη του πληθυσμού μιας οντότητας A συσχετίζονται με/αντιστοιχούν σε ένα ή περισσότερα μέλη του πληθυσμού μιας οντότητας B.

Στο Σχήμα 4.15 φαίνεται ένα παράδειγμα μιας συσχέτισης 1:N μεταξύ των οντοτήτων «καθηγητής» και «μάθημα». Ο αναγνώστης καλείται να προσέξει ότι η συσχέτιση χαρακτηρίζεται από την τιμή των πεδίων «αριθμός ταυτότητας» και «διδάσκων» η οποία είναι η ίδια στις συσχετιζόμενες εγγραφές στους δύο πίνακες. Στο Σχήμα 4.16 φαίνεται η υλοποίηση των συσχετίσεων 1:1 και M:N. Σχετικά με τις συσχετίσεις M:N μπορεί κανείς να παρατηρήσει ότι μια τέτοια συσχέτιση μεταξύ των πινάκων A και B μπορεί να θεωρηθεί ως μια συσχέτιση 1:N από τον πίνακα A στον πίνακα B και, ταυτόχρονα, ισχύει το αντίστροφο, δηλαδή είναι και μια συσχέτιση 1:N από τον πίνακα B στον πίνακα A.

**Σχήμα 4.15** Μια συσχέτιση 1:N μεταξύ των οντοτήτων «καθηγητής» και «μάθημα».


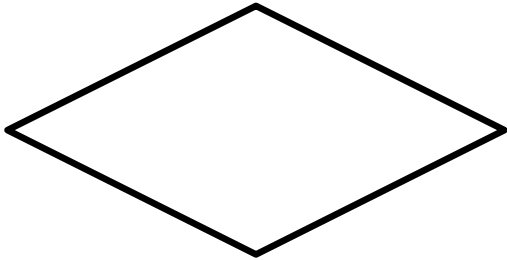
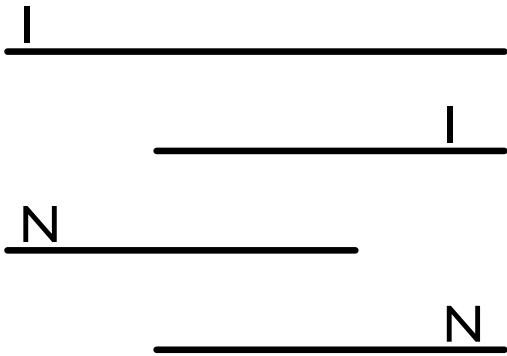
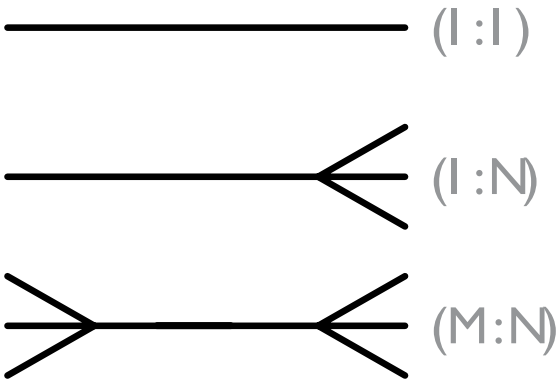


**Σχήμα 4.16** Συσχετίσεις 1:1 και M:N στο σχεσιακό μοντέλο δεδομένων.



Με τη βοήθεια του διαγράμματος οντοτήτων – συσχετίσεων καταγράφονται στη φάση της προδιαγραφής των απαιτήσεων από το λογισμικό οι απαιτήσεις σε μόνιμη αποθήκευση δεδομένων, χωρίς να ενδιαφέρει ιδιαίτερα η κατασκευαστική λεπτομέρεια. Οι συμβολισμοί που χρησιμοποιούνται στα διαγράμματα οντοτήτων – συσχετίσεων φαίνονται στο Σχήμα 4.17.

**Σχήμα 4.17** Συμβολισμοί διαγραμμάτων οντοτήτων – συσχετίσεων.

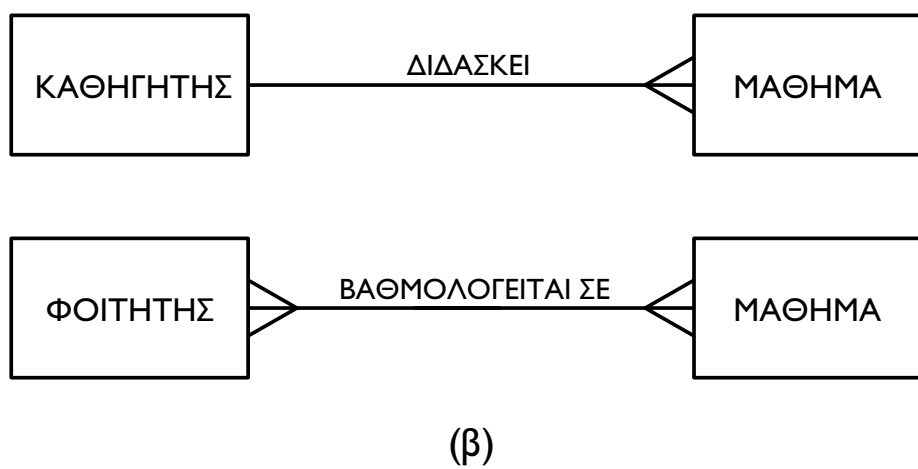
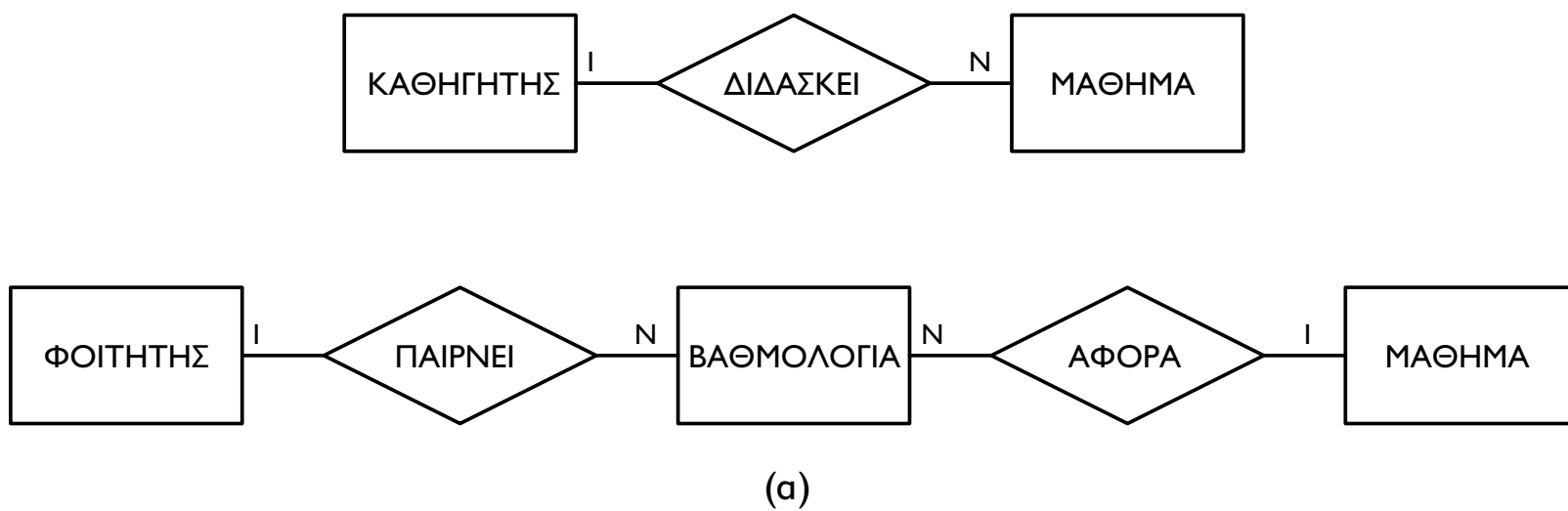
Συμβολισμοί διαγραμμάτων οντοτήτων - συσχετίσεων	
	Οντότητα δεδομένων
	Συσχέτιση μεταξύ οντοτήτων (α)
	Ορισμός πολλαπλότητας συσχέτισης(α)
	Συσχέτιση και ορισμός πολλαπλότητας (β)

Κάθε οντότητα περιγράφεται με χρήση ενός παραλληλογράμμου, μέσα στο οποίο σημειώνεται το όνομά της. Για την παράσταση των συσχετίσεων, στο σχήμα φαίνονται δύο εναλλακτικοί τρόποι.

- Πρώτον, με τη βοήθεια ενός ρόμβου μέσα στον οποίο σημειώνεται το ρήμα που τις χαρακτηρίζει και στις ακμές η πολλαπλότητα της συσχέτισης. Αυτός είναι και ο συμβολισμός που χρησιμοποιήθηκε αρχικά στο διάγραμμα οντοτήτων – συσχετίσεων και αναφέρεται εδώ μόνο για ιστορικούς λόγους.
- Δεύτερον, με τη βοήθεια γραμμών που συνδέουν τις συσχετιζόμενες οντότητες ανάλογα με την κατάληξη των οποίων δηλώνεται το είδος της συσχέτισης. Το όνομα της συσχέτισης σημειώνεται επάνω από τη γραμμή σε κάποιο πρόσφορο σημείο. Πρόκειται για τον συμβολισμό που θα χρησιμοποιηθεί στη συνέχεια του βιβλίου αυτού.

Στο Σχήμα 4.18 φαίνεται ένα παράδειγμα διαγράμματος οντοτήτων – συσχετίσεων σύμφωνα και με τους δύο συμβολισμούς.

**Σχήμα 4.18** Ένα παράδειγμα διαγράμματος οντοτήτων – συσχετίσεων.





Μια συσχέτιση δεν είναι αντιμεταθετική, δηλαδή δεν μπορεί να διαβαστεί με την αντίθετη φορά με το ίδιο ρήμα. Στο παράδειγμα του προηγούμενου σχήματος οι συσχετίσεις διαβάζονται ως εξής:

Ευθέως:

*Κάθε καθηγητής διδάσκει πολλά μαθήματα.*

*Κάθε φοιτητής βαθμολογείται σε κάθε μάθημα πολλές φορές.*

Αντίστροφα:

*Κάθε μάθημα διδάσκεται από έναν καθηγητή.*

*Σε κάθε μάθημα, κάθε φοιτητής λαμβάνει τουλάχιστον μία βαθμολογία.*

Αξίζει να σημειωθεί ότι η μετατροπή της σύνταξης από ενεργητική σε παθητική δεν είναι πάντα επαρκής για την ανάγνωση μιας συσχέτισης με την αντίστροφη φορά. Επίσης, θα πρέπει να εντοπιστεί η ύπαρξη της οντότητας «βαθμολογία» στο Σχήμα 4.18 (α), η οποία μετατρέπει τη συσχέτιση M:N μεταξύ των οντοτήτων «φοιτητής» και «μάθημα» σε δύο συσχετίσεις I:N.

Όπως ήδη αναφέρθηκε, η σχεδίαση τέτοιων διαγραμμάτων αποτελεί μέρος της σχεδίασης σχεσιακών βάσεων δεδομένων, η οποία αποτελεί αυτοτελές γνωστικό αντικείμενο. Χωρίς να είναι δυνατό να υπεισέλθουμε σε βάθος, θα παραθέσουμε ακολούθως ορισμένα σημεία που είναι χρήσιμο να λαμβάνονται υπόψη στη σχεδίαση διαγραμμάτων οντοτήτων – συσχετίσεων στη φάση της προδιαγραφής των απαιτήσεων από το λογισμικό.

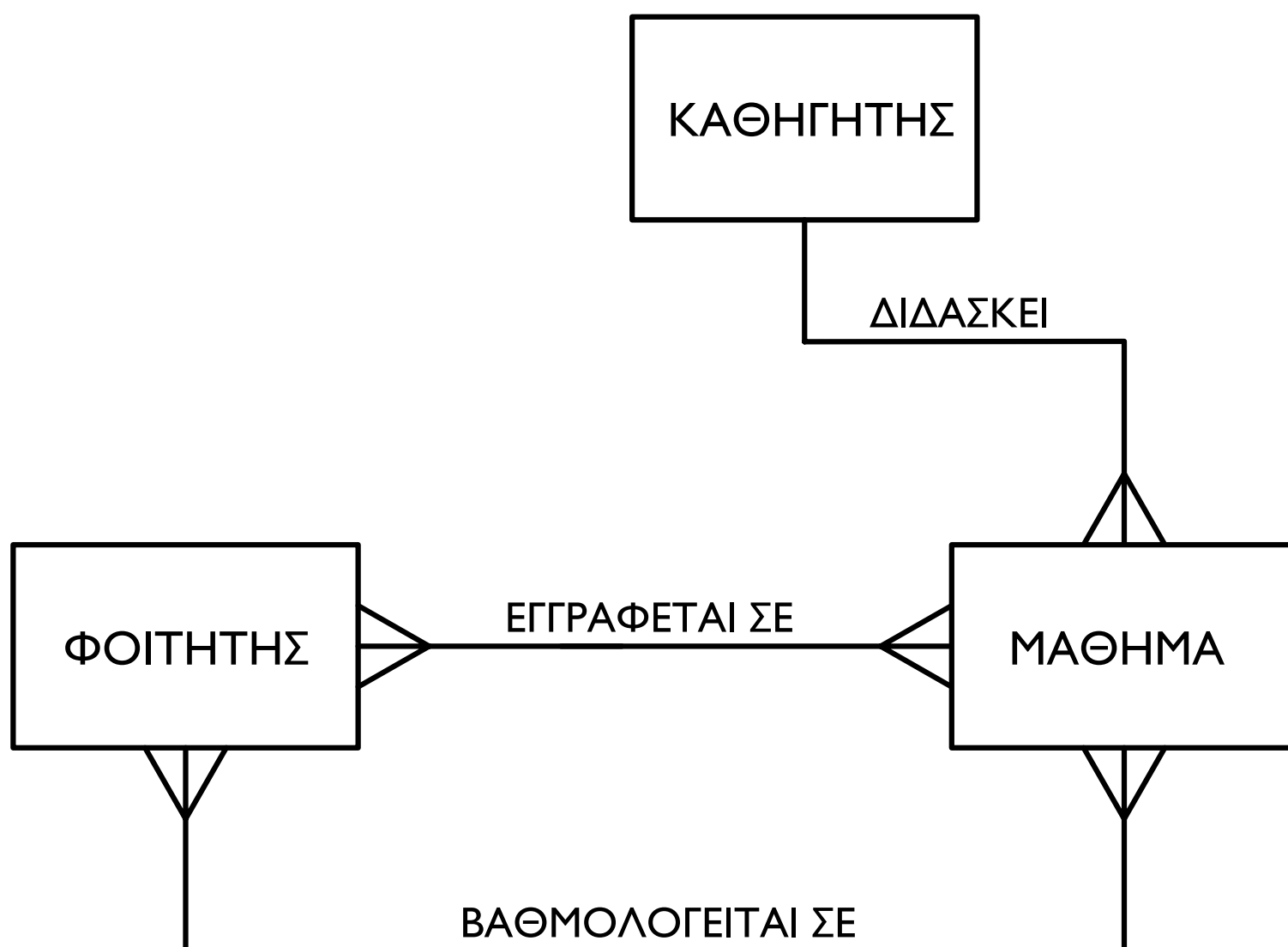
- Σκοπός είναι ο καταρχήν προσδιορισμός των οντοτήτων και των συσχετίσεων μεταξύ τους. Οι λεπτομέρειες που αφορούν κάθε οντότητα και συσχέτιση, όπως το πλήθος και το είδος των πεδίων, θα προσδιοριστούν πλήρως σε επόμενη φάση της ανάπτυξης του λογισμικού.
- Η απλότητα και η εύκολη αναγνωσιμότητα του διαγράμματος θα πρέπει να αποτελεί επιδίωξη. Όσο πιο σύνθετο είναι ένα διάγραμμα οντοτήτων – συσχετίσεων, τόσο πιο δύσκολη είναι η υλοποίησή του και οι μεταβολές σε αυτό στη συνέχεια, ενώ συνήθως υπάρχει πάντα εναλλακτικός τρόπος σχεδίασης.

- Δεν είναι σκόπιμο να γίνει σε αυτή τη φάση καμία τροποποίηση με στόχο τη βελτιστοποίηση. Τέτοιες ενέργειες μπορούν να γίνουν στη συνέχεια, κατά τη φάση της σχεδίασης του λογισμικού.
- Η σχεδίαση του διαγράμματος οντοτήτων – συσχετίσεων στην παρούσα φάση δεν θα πρέπει να γίνεται έχοντας κατά νου κάποιο συγκεκριμένο περιβάλλον υλοποίησης (σύστημα διαχείρισης βάσεων δεδομένων, γλώσσα προγραμματισμού κ.λπ.). Τα πλεονεκτήματα (και τα μειονεκτήματα) του συγκεκριμένου περιβάλλοντος υλοποίησης θα απασχολήσουν και αυτά τον κατασκευαστή κατά τη φάση της σχεδίασης του λογισμικού, η οποία έπεται.

## **Μελέτη περίπτωσης/Κεφάλαιο 4**

Στο Σχήμα 4.19 φαίνεται το διάγραμμα οντοτήτων – συσχετίσεων του λογισμικού «Επίκουρος», όπως μπορεί να διαμορφωθεί σύμφωνα με όσα έχουν αναφερθεί μέχρι το σημείο αυτό.

**Σχήμα 4.19** Το διάγραμμα οντοτήτων – συσχετίσεων της εφαρμογής λογισμικού «Επίκουρος».



Στο διάγραμμα εντοπίζουμε τις τρεις οντότητες «φοιτητής», «καθηγητής» και «μάθημα». Μεταξύ της οντότητας «καθηγητής» και «μάθημα» εντοπίζουμε τη συσχέτιση «διδάσκει». Η περιγραφή της συσχέτισης με αυτό τον τρόπο υποκρύπτει την παραδοχή ότι κάθε μάθημα διδάσκεται μόνο από έναν καθηγητή, η οποία δεν ανταποκρίνεται απαραίτητα στην πραγματικότητα ενός εκπαιδευτικού οργανισμού, χωρίς αυτό να επηρεάζει τους σκοπούς του παραδείγματός μας. Μεταξύ των οντοτήτων «φοιτητής» και «μάθημα» εντοπίζουμε δύο συσχετίσεις: αυτή που περιγράφει την εγγραφή και αυτή που περιγράφει τη βαθμολογία. Και οι δύο είναι συσχετίσεις M:N διότι μπορούν να αναγνωστούν ως συσχετίσεις I:N και από τις δύο κατευθύνσεις.

#### Άσκηση 4/Κεφάλαιο 4

Ελέγξτε τη συνέπεια του διαγράμματος οντοτήτων – συσχετίσεων της εφαρμογής «Επίκουρος» με τα εικονιζόμενα στα διαγράμματα ροής δεδομένων. Εντοπίστε ένα σημείο ασυνέπειας, δηλαδή έννοιας που αφορά οντότητα δεδομένων και εμφανίζεται στο διάγραμμα ροής δεδομένων, ενώ δεν εμφανίζεται στο διάγραμμα οντοτήτων – συσχετίσεων. Προσπαθήστε να εξηγήσετε την ασυνέπεια αυτή.

#### Άσκηση 5/Κεφάλαιο 4

Προσπαθήστε να αναγνώσετε τις δύο M:N συσχετίσεις μεταξύ των οντοτήτων «φοιτητής» και «μάθημα» και από τις δύο κατευθύνσεις ως σχέσεις I:N.

#### 4.4.4. Διαγράμματα μετάβασης καταστάσεων

Όπως αναφέρθηκε σε προηγούμενη παράγραφο, με τα μοντέλα παράστασης λογισμικού που έχουμε παρουσιάσει μέχρι τώρα δεν παριστάνεται η δυναμική συμπεριφορά του λογισμικού, δηλαδή η χρονική σειρά εκτέλεσης εργασιών ανάλογα με τα εξωτερικά γεγονότα τα οποία προκαλεί ο χρήστης

ή άλλες εξωτερικές πηγές. Η περιγραφή της συμπεριφοράς αυτής είναι απαραίτητη, προκειμένου να αποκτήσει κανείς μια καλή εικόνα της εφαρμογής λογισμικού που προδιαγράφεται. Ένα χρήσιμο για το σκοπό αυτό εργαλείο είναι το διάγραμμα μετάβασης καταστάσεων (state transition diagram).

Για να μιλήσουμε με όρους που έχουμε ήδη χρησιμοποιήσει, αυτό που ζητείται κατά τον προσδιορισμό της δυναμικής συμπεριφοράς του λογισμικού είναι η σειρά εφαρμογής των μετασχηματισμών που περιγράφονται στο διάγραμμα ροής δεδομένων. Επαναλαμβάνουμε ότι η σειρά αυτή δεν περιγράφεται στο διάγραμμα ροής δεδομένων αλλά ελέγχεται από παράγοντες όπως είναι ο χρήστης ή εξωτερικά προς την εφαρμογή λογισμικού συμβάντα. Ανάλογα με το μήνυμα που λαμβάνει το λογισμικό από το περιβάλλον του, εκτελεί μια συγκεκριμένη εργασία. Ένα απλό παράδειγμα είναι ένας κεντρικός πίνακας επιλογής εργασιών (μενού) όπου, ανάλογα με την επιλογή του χρήστη, μπορεί να εκτελείται μια εργασία, να εμφανίζεται ένα νέο μενού ή να τερματίζεται η εκτέλεση του προγράμματος.

Είναι αναμενόμενο ότι δεν είναι δυνατή η εκτέλεση οποιασδήποτε εργασίας σε οποιαδήποτε στιγμή. Ανάλογα με την κατάσταση που βρίσκεται το σύστημα και με το μήνυμα που λαμβάνει, εκτελείται μια εργασία και το σύστημα μεταβαίνει ενδεχομένως σε μια άλλη κατάσταση. Αυτή ακριβώς η συμπεριφορά είναι που περιγράφεται με το διάγραμμα μετάβασης καταστάσεων. Είναι σκόπιμο να παραθέσουμε μερικούς ορισμούς.

### **Γεγονός:**

Ένα γεγονός (event) είναι μια στιγμιαία μεταβολή στο περιβάλλον λειτουργίας του λογισμικού η οποία προκαλείται από εξωτερικούς παράγοντες (χρήστης, λειτουργικό σύστημα, άλλες εφαρμογές λογισμικού).

### **Απόκριση:**






Μια λειτουργία που εκτελεί το λογισμικό όταν προκαλείται ένα γεγονός ονομάζεται «απόκριση» (response). Στη δομημένη ανάλυση και σχεδίαση δεν είναι όλες οι λειτουργίες εκτελέσιμες ως αποκρίσεις σε γεγονότα.

### **Κατάσταση:**

Όταν το λογισμικό αναμένει («αφουγκράζεται») γεγονότα, τότε λέμε ότι βρίσκεται σε μία κατάσταση. Με τη λήψη ενός γεγονότος, το λογισμικό μπορεί να εκτελεί μια λειτουργία και να μεταβαίνει σε μια άλλη κατάσταση.

Σε κάθε κατάσταση είναι καθορισμένα τα γεγονότα τα οποία μπορούν να προκαλέσουν μετάβαση και πρέπει να είναι σαφές ποιο γεγονός προκαλεί μετάβαση σε ποια νέα κατάσταση, καθώς και το ποια λειτουργία εκτελείται. Σε κάθε περίπτωση υπάρχουν δύο καταστάσεις που χαρακτηρίζονται ως καταστάσεις έναρξης και τέλους. Όλα αυτά καθορίζονται από το διάγραμμα μετάβασης καταστάσεων, το οποίο μπορεί να αφορά ολόκληρο το λογισμικό ή οποιοδήποτε υποσύστημά του. Οι συμβολισμοί που χρησιμοποιούνται στη σχεδίαση του διαγράμματος μετάβασης καταστάσεων φαίνονται στο Σχήμα 4.20.

**Σχήμα 4.20** Συμβολισμοί διαγραμμάτων μετάβασης καταστάσεων.

Συμβολισμοί διαγραμμάτων μετάβασης καταστάσεων	
	Κατάσταση
	Κατάσταση έναρξης
	Κατάσταση τέλους
	Μετάβαση σε άλλη κατάσταση / λειτουργία που εκτελείται
	Μετάβαση στην ίδια κατάσταση / λειτουργία που εκτελείται

Στο διάγραμμα μετάβασης καταστάσεων, οι καταστάσεις παριστάνονται με ένα παραλληλόγραμμο με στρογγυλεμένες γωνίες. Οι καταστάσεις έναρξης και τέλους παριστάνονται με μια μεγάλη μαύρη κουκίδα εκτός ή εντός κύκλου αντίστοιχα. Οι μεταβάσεις μεταξύ καταστάσεων περιγράφονται με ένα βέλος από την αρχική στην τελική κατάσταση. Πάνω στο βέλος σημειώνεται το γεγονός που προκαλεί τη μετάβαση και, προαιρετικά, η απόκριση στο γεγονός αυτό. Το να συμπεριλάβει κανείς την πληροφορία της απόκρισης δεν είναι πάντα εύκολο για λόγους αναγνωσιμότητας του διαγράμματος.

Δεν είναι εύκολη η απάντηση στο ερώτημα πόσα διαγράμματα μετάβασης καταστάσεων είναι αναγκαίο να κατασκευαστούν στη φάση της προδιαγραφής των απαιτήσεων από το λογισμικό. Στη δομημένη ανάλυση και σχεδίαση το λογισμικό θεωρείται ως ένα σύνολο από λειτουργίες, η ελεγχόμενη εκτέλεση των οποίων φέρει το επιθυμητό αποτέλεσμα. Ως εκ τούτου, είναι χρήσιμο να κατασκευάζεται τουλάχιστον ένα τέτοιο διάγραμμα που να περιγράφει τον τρόπο εκκίνησης της εκτέλεσης των εργασιών, ο οποίος συνήθως είναι ένα σύνολο από πίνακες επιλογών (μενού).

Για ορισμένες από τις εργασίες και, κυρίως, για εκείνες των οποίων η ροή της εκτέλεσης δεν είναι εύκολα ή με μοναδικό τρόπο αντιληπτή, είναι χρήσιμο επίσης το διάγραμμα μετάβασης καταστάσεων. Ένας θεωρητικός αντίλογος στην παραπάνω διατύπωση μπορεί να απαιτεί την κατασκευή τέτοιων διαγραμμάτων για όλες τις εργασίες. Η εμπειρία έχει δείξει ότι στην πράξη κάτι τέτοιο όχι μόνο δεν είναι χρήσιμο αλλά, πέραν από κόπο, ενδεχομένως να προσθέτει και σύγχυση. Ακολουθώντας παραθέτουμε δύο απλά πρακτικά κριτήρια για να αποφασίζει κανείς πότε θα κατασκευάσει ένα διάγραμμα μετάβασης καταστάσεων.

- Όταν η περιγραφή της συμπεριφοράς μιας μονάδας λογισμικού με ένα τέτοιο διάγραμμα βοηθάει στην καλύτερη κατανόησή της και (όπως θα δούμε στη συνέχεια) στη σχεδίαση του αντίστοιχου πηγαίου κώδικα.
- Όταν πρόκειται να περιγράψουμε τη συμπεριφορά μιας μονάδας η οποία σχετίζεται με τη διαχείριση δεδομένων που αφορούν γεγονότα του πραγματικού κόσμου.

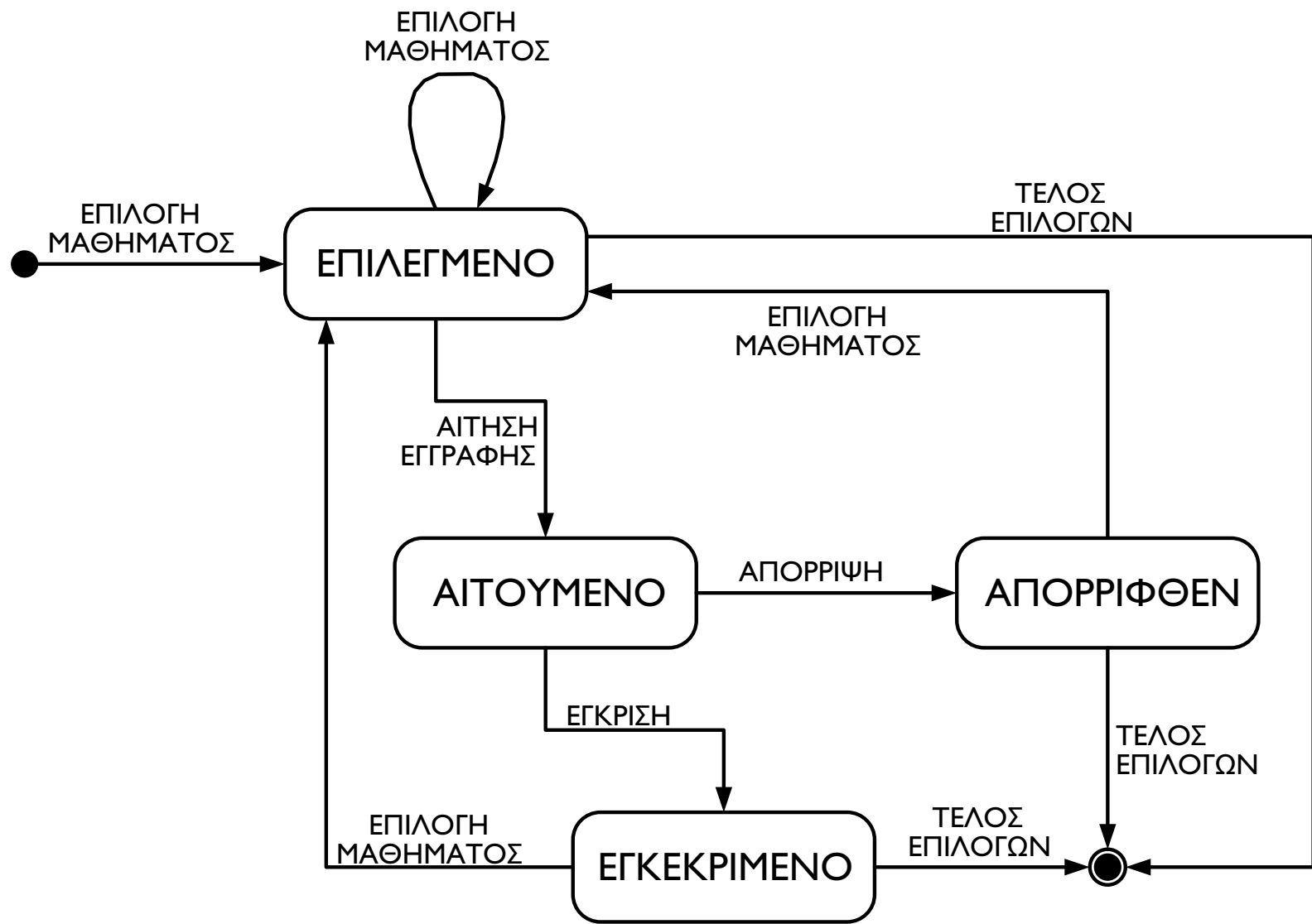


## Μελέτη περίπτωσης/Κεφάλαιο 4

Στο Σχήμα 4.21 φαίνεται ένα διάγραμμα μετάβασης καταστάσεων από την εφαρμογή λογισμικού «Επίκουρος». Το διάγραμμα περιγράφει τη συμπεριφορά της μονάδας εκείνης του λογισμικού η οποία είναι υπεύθυνη για την εγγραφή φοιτητών σε μαθήματα.

Η μονάδα πρέπει να επιβεβαιώνει ότι ο φοιτητής έχει τις προϋποθέσεις για να εγγραφεί σε ένα μάθημα, όπως αυτές ορίζονται με τη μορφή προαπαιτήσεων, εξαμήνου ή άλλες. Με το ξεκίνημα της λειτουργίας προσφέρεται η δυνατότητα επιλογής μαθήματος, για παράδειγμα μέσα από μια λίστα. Με την «αίτηση εγγραφής», η οποία μπορεί να γίνεται με το πάτημα ενός κουμπιού, το μάθημα χαρακτηρίζεται ως αιτούμενο. Από την κατάσταση αυτή μπορεί να προκύψουν δύο γεγονότα: η «απόρριψη» (λόγω, για παράδειγμα, προαπαιτήσεων), οπότε το μάθημα χαρακτηρίζεται ως απορριφθέν και η επιλογή μπορεί να συνεχιστεί, ή η έγκριση, οπότε το μάθημα χαρακτηρίζεται εγκεκριμένο, η εγγραφή γίνεται και η επιλογή μπορεί να συνεχιστεί. Σε οποιαδήποτε στιγμή, η πρόκληση του γεγονότος «τέλος επιλογών» από τον χρήστη επιφέρει μετάβαση στην τελική κατάσταση.

**Σχήμα 4.21** Ένα παράδειγμα διαγράμματος μετάβασης καταστάσεων.



## Δραστηριότητα 9/Κεφάλαιο 4

Στη μελέτη περίπτωσης του παρόντος κεφαλαίου ένα μάθημα μπορεί να διέρχεται από τις καταστάσεις «νέο», «σε τρέχουσα διδασκαλία», «αρχείο». Τα γεγονότα που αφορούν τις μεταβάσεις μεταξύ των καταστάσεων αυτών είναι τα «καταχώρηση νέου μαθήματος», «ορισμός διδάσκοντα», «ενημέρωση βαθμολογίας», «εγγραφή φοιτητή» και «αρχειοθέτηση». Κατασκευάστε ένα διάγραμμα μετάβασης καταστάσεων που περιέχει τα παραπάνω.

### 4.4.5. Το λεξικό δεδομένων

Στη μέχρι τώρα αναφορά μας στα μοντέλα παράστασης λογισμικού έχουμε εισάγει ένα σύνολο από έννοιες και συμβολισμούς με τη βοήθεια των οποίων παριστάνουμε τις απαιτήσεις μας από το λογισμικό ιδωμένες από διαφορετικές οπτικές γωνίες. Οι έννοιες αυτές αφορούν τρία πράγματα: λειτουργίες, δεδομένα και έλεγχο, και περιγράφονται αντίστοιχα με τα διαγράμματα ροής δεδομένων, οντοτήτων – συσχετίσεων και μετάβασης καταστάσεων.

Αναφέρθηκε στην αρχή της παρούσας ενότητας ότι τα περιεχόμενα των διαγραμμάτων αυτών οφείλουν να είναι συνεπή μεταξύ τους, δηλαδή η χρήση της ίδιας ονομασίας σε διαφορετικά διαγράμματα να αναφέρεται στην ίδια έννοια. Αν λάβει κανείς υπόψη το γεγονός ότι βρισκόμαστε ακόμη σε σχετικά πρόωρο στάδιο της ανάπτυξης λογισμικού, όπου η δημιουργία συγχύσεων είναι μάλλον εύκολη, αλλά και το γεγονός ότι η πολυπλοκότητα στη σχεδίαση ενός διαγράμματος μπορεί να μην επιτρέπει να περιγράφονται με κατατοπιστικούς τίτλους τα όσα φαίνονται σε αυτό, καταλαβαίνει εύκολα την αναγκαιότητα ενός πιο οργανωμένου τρόπου περιγραφής των εννοιών που περιέχονται στα μοντέλα παράστασης λογισμικού. Για τον σκοπό αυτό χρησιμοποιείται ένας πίνακας που ονομάζεται «λεξικό δεδομένων» (data dictionary).

## Λεξικό δεδομένων:

Το λεξικό δεδομένων είναι μια οργανωμένη ταξινόμηση όλων των σχετιζόμενων με δεδομένα στοιχείων των μοντέλων παράστασης λογισμικού με όσο το δυνατόν μεγαλύτερη σαφήνεια και πληρότητα, έτσι ώστε να γίνονται κατανοητά τόσο από τον αναλυτή του συστήματος, όσο και από τον χρήστη αυτού.

Δεν είναι δύσκολο να καταλάβει κανείς ότι η δημιουργία και συνεχής ενημέρωση ενός λεξικού δεδομένων είναι μια αρκετά δύσκολη, χρονοβόρα και επιρρεπής σε σφάλματα εργασία. Συνήθως το λεξικό δεδομένων δημιουργείται αυτόματα από τα εργαλεία υποστήριξης της ανάπτυξης του λογισμικού (CASE tools), τα οποία καθορίζουν και τα ακριβή περιεχόμενά του. Μια συνηθισμένη δομή ενός λεξικού δεδομένων είναι ένας πίνακας που περιλαμβάνει τα παρακάτω πεδία:

- **Ονομασία:** Το κύριο αναγνωριστικό της οντότητας, του πεδίου ή της ροής δεδομένων.
- **Βοηθητικές ονομασίες:** Δευτερεύουσες ονομασίες που χρησιμοποιούνται χάρη συντομίας ή ισοδύναμα.
- **Πού χρησιμοποιείται:** Αναφορά στους μετασχηματισμούς, οντότητες κ.λπ. οι οποίοι χρησιμοποιούν το εν λόγω στοιχείο.
- **Πώς χρησιμοποιείται:** Αναφορά στον τρόπο με τον οποίο χρησιμοποιείται το εν λόγω στοιχείο (ως στοιχείο εισόδου, ως αποτέλεσμα, πεδίο κ.ά.).
- **Τι περιέχει:** Περιγραφή του είδους και της μορφής της πληροφορίας που αποθηκεύεται σε αυτό.
- **Όρια τιμών:** Καθορισμός των επιτρεπτών τιμών που μπορεί να πάρει (αν απαιτείται).
- **Αρχική τιμή:** Καθορισμός της αρχικής τιμής του στοιχείου (αν απαιτείται).
- **Λοιπά στοιχεία:** Υπόλοιπες χρήσιμες πληροφορίες.

Στη γενικότερη και πληρέστερη περίπτωση, το λεξικό δεδομένων είναι το ίδιο μια βάση δεδομένων στην οποία αποθηκεύονται περισσότερες πληροφορίες από τις παραπάνω, όπως χαρακτηριστικά η αποσύνθεση σύνθετων δεδομένων σε απλούστερα η οποία εμφανίζεται στα διαγράμματα ροής δεδομένων. Η κατασκευή του λεξικού δεδομένων δεν είναι μια ανεξάρτητη εργασία αλλά γίνεται παράλληλα με την κατασκευή των μοντέλων παράστασης λογισμικού. Με την πρώτη εμφάνιση ενός στοιχείου δεδομένων, αυτό εισάγεται στο λεξικό και χαρακτηρίζεται όσο πληρέστερα γίνεται. Ο χαρακτηρισμός του ολοκληρώνεται καθώς προχωρά η ανάπτυξη.

Θεωρητικά, το λεξικό δεδομένων αποτελεί τον ακρογωνιαίο λίθο πάνω στον οποίο βασίζεται η συνέπεια μεταξύ των μοντέλων παράστασης λογισμικού, όπως φαίνεται στο Σχήμα 4.6. Πρακτικά, η κατασκευή και ενημέρωσή του με το χέρι είναι μια πολύ δύσκολη εργασία, η συχνότερη κατάληξη της οποίας είναι το λεξικό δεδομένων να μένει ημιτελές, μη ενημερωμένο και μετά από λίγο να μην είναι χρήσιμο στον κατασκευαστή. Τα λεξικά δεδομένων έχουν εξελιχθεί σε αποθήκες πληροφοριών λογισμικού (software repositories) και, εκτός από δεδομένα, περιλαμβάνουν και ενεργά συστατικά στοιχεία λογισμικού τα οποία μπορεί να επαναχρησιμοποιηθούν. Η χρήση εργαλείων υποστήριξης της ανάπτυξης του λογισμικού (CASE) καθιστά αρκετά ευκολότερη την εργασία δημιουργίας και συντήρησης ενός λεξικού δεδομένων.

#### **Μελέτη περίπτωσης/Κεφάλαιο 4**

Στον Πίνακα 4.1 φαίνεται το τμήμα του λεξικού δεδομένων που αντιστοιχεί στα δεδομένα που περιλαμβάνονται στο διάγραμμα οντοτήτων – συσχετίσεων που εικονίζεται στο Σχήμα 4.18. Το λεξικό περιέχει αναφορές στους πίνακες του διαγράμματος οντοτήτων – συσχετίσεων, καθώς και την περιγραφή ορισμένων πεδίων κάθε πίνακα.

ΟΝΟΜΑΣΙΑ	ΑΛΛΕΣ ΟΝΟΜΑΣΙΕΣ	ΠΟΥ	ΠΩΣ	ΠΕΡΙΕΧΟΜΕΝΑ	ΟΡΙΑ
ΚΑΘΗΓΗΤΗΣ	ΚΑΘ	Βάση Δεδομένων	Πίνακας της ΒΔ	Εγγραφές καθηγητών	-
ΜΑΘΗΜΑ	ΜΑΘ	Βάση Δεδομένων	Πίνακας της ΒΔ	Εγγραφές μαθημάτων	-
ΒΑΘΜΟΛΟΓΙΑ	ΒΑΘΜ	Βάση Δεδομένων	Πίνακας της ΒΔ	Εγγραφές βαθμολογίας	-
ΦΟΙΤΗΤΗΣ	ΦΟΙΤ	Βάση Δεδομένων	Πίνακας της ΒΔ	Εγγραφές φοιτητών	-
ΑΡ. ΤΑΥΤΟΤΗΤΑΣ	Α.Τ.	Πίνακας «ΚΑΘΗΓΗΤΗΣ»	Πεδίο	Πεδίο 7 χαρακτήρων	-
ΟΝΟΜΑ	ΟΝ	Πίνακας «ΚΑΘΗΓΗΤΗΣ»	Πεδίο	Πεδίο 25 χαρακτήρων	-
ΕΠΩΝΥΜΟ	ΕΠ	Πίνακας «ΚΑΘΗΓΗΤΗΣ»	Πεδίο	Πεδίο 25 χαρακτήρων	-
ΔΙΕΥΘΥΝΣΗ	ΔΙΕΥΘ	Πίνακας «ΚΑΘΗΓΗΤΗΣ»	Πεδίο	Πεδίο 50 χαρακτήρων	-
ΤΗΛΕΦΩΝΟ	ΤΗΛ	Πίνακας «ΚΑΘΗΓΗΤΗΣ»	Πεδίο	Πεδίο 20 χαρακτήρων	-
ΑΡ. ΤΑΥΤΟΤΗΤΑΣ	Α.Τ.	Πίνακας «ΦΟΙΤΗΤΗΣ»	Πεδίο	Πεδίο 7 χαρακτήρων	-
ΟΝΟΜΑ	ΟΝ.Φ.	Πίνακας «ΦΟΙΤΗΤΗΣ»	Πεδίο	Πεδίο 25 χαρακτήρων	-
ΕΠΩΝΥΜΟ	ΕΠΦ.	Πίνακας «ΦΟΙΤΗΤΗΣ»	Πεδίο	Πεδίο 25 χαρακτήρων	-
ΔΙΕΥΘΥΝΣΗ	ΔΙΕΥΘ.Φ.	Πίνακας «ΦΟΙΤΗΤΗΣ «	Πεδίο	Πεδίο 50 χαρακτήρων	-
ΤΗΛΕΦΩΝΟ	ΤΗΛ.Φ.	Πίνακας «ΦΟΙΤΗΤΗΣ»	Πεδίο	Πεδίο 20 χαρακτήρων	-
ΤΜΗΜΑ	ΤΜ.Φ.	Πίνακας «ΦΟΙΤΗΤΗΣ «	Πεδίο	Πεδίο 20 χαρακτήρων	-
ΚΩΔΙΚΟΣ ΜΑΘ	Κ.Μ.	Πίνακας «ΜΑΘΗΜΑ»	Πεδίο	Πεδίο 10 αριθμητικών ψηφίων	-
ΘΕΜΑΤΙΚΗ ΕΝΟΤΗΤΑ	Θ.ΕΝ.	Πίνακας «ΜΑΘΗΜΑ»	Πεδίο	Πεδίο 10 χαρακτήρων	-
ΤΙΤΛΟΣ	-	Πίνακας «ΜΑΘΗΜΑ»	Πεδίο	Πεδίο 50 χαρακτήρων	-
ΔΙΔΑΣΚΩΝ	ΔΙΔ	Πίνακας «ΜΑΘΗΜΑ»	Πεδίο	Πεδίο 7 χαρακτήρων	-
ΗΜΕΡΟΜΗΝΙΑ ΕΞΕΤΑΣΗΣ	ΗΜ.ΕΞ.	Πίνακας «ΒΑΘΜΟΛΟΓΙΑ»	Πεδίο	Πεδίο ημερομηνίας (ΗΗ/ΜΜ/ΕΕΕΕ)	< τρέχουσα ημερομηνία
ΕΙΔΟΣ ΕΞΕΤΑΣΗΣ	ΕΙΔ.ΕΞ.	Πίνακας «ΒΑΘΜΟΛΟΓΙΑ»	Πεδίο	Πεδίο 10 χαρακτήρων	Ενδιάμεση, Τελική
ΒΑΘΜΟΣ	ΒΑΘΜ.	Πίνακας «ΒΑΘΜΟΛΟΓΙΑ»	Πεδίο	Αριθμητικό πεδίο	0-10
ΠΑΡΑΤΗΡΗΣΕΙΣ	-	Πίνακας «ΒΑΘΜΟΛΟΓΙΑ»	Πεδίο	Πεδίο 100 χαρακτήρων	-

**Πίνακας 4.1** Τμήμα του λεξικού δεδομένων που αντιστοιχεί στο διάγραμμα οντοτήτων – συσχετίσεων που εικονίζεται στο Σχήμα 4.18. Για οικονομία χώρου έχουν παραληφθεί οι στήλες «αρχική τιμή» και «παρατηρήσεις».



## ΕΝΟΤΗΤΑ 4.5. ΠΡΟΒΛΗΜΑΤΑ ΣΤΟΝ ΠΡΟΣΔΙΟΡΙΣΜΟ ΑΠΑΙΤΗΣΕΩΝ

Κατά τη φάση της προδιαγραφής των απαιτήσεων από το λογισμικό επιδιώκεται να καθοριστεί το *τι* θα κάνει μια υπό κατασκευή εφαρμογή λογισμικού, καθώς και ποια γενικά χαρακτηριστικά θα έχει. Στη διαδικασία αυτή εμπλέκονται τόσο ο κατασκευαστής, όσο και ο πελάτης. Το αποτέλεσμα δεν είναι πάντα το επιθυμητό εξαιτίας διαφόρων προβλημάτων που παρατηρούνται, μια σύντομη αναφορά σε κάποια από τα οποία θα επιχειρήσουμε στην ενότητα αυτή.

Η φύση του λογισμικού είναι τέτοια που εντείνει τα προβλήματα επικοινωνίας που ούτως ή άλλως υπάρχουν μεταξύ ατόμων με διαφορετική κουλτούρα. Από την πλευρά των κατασκευαστών και της ερευνητικής κοινότητας που ασχολείται με το λογισμικό δεν έχει καταστεί δυνατό να υπάρξει συμφωνία σχετικά με τις έννοιες και τους συμβολισμούς που χρησιμοποιούνται στην περιγραφή του λογισμικού. Σε αυτά προστίθεται η φυσική γλώσσα η οποία συχνά περιέχει διφορούμενα και ασάφειες αλλά και η υπόσταση του λογισμικού ως προϊόντος που κατασκευάζεται με σκοπό το κέρδος. Μπορούμε, λοιπόν, να διακρίνουμε τέσσερις κατηγορίες προβλημάτων που απαντώνται στην προδιαγραφή των απαιτήσεων από το λογισμικό: επικοινωνίας, προτύπων, γλώσσας και οικονομικά.

### 4.5.1. Προβλήματα επικοινωνίας

Όταν ένας πελάτης συνεργάζεται με έναν κατασκευαστή για να του περιγράψει τις απαιτήσεις του από ένα τεχνικό έργο, είναι σημαντικό να έχουν αμφότεροι τις ίδιες παραστάσεις σχετικά με το έργο αυτό, να μπορούν δηλαδή να μιλήσουν την ίδια γλώσσα. Στην περίπτωση, λόγου χάρη, ενός οικοδομικού έργου, είναι αρκετά εύκολο ο πελάτης να περιγράψει ακριβώς αυτό που θέλει και ο κατασκευαστής να καταλάβει το ίδιο πράγμα.

Δεν ισχύει δυστυχώς το ίδιο με το λογισμικό. Συχνά ο πελάτης αδυνατεί όχι μόνο να περιγράψει αλλά ακόμα και να αντιληφθεί αυτό που πραγματικά θέλει κυριαρχούμενος από την εντύπωσή του για το λογισμικό και την πληροφορική γενικά, η οποία μπορεί να απέχει αρκετά από την πραγματικότητα.

Στην περίπτωση αυτή ο κατασκευαστής δεν είναι απλά ο καταγραφέας των απαιτήσεων αλλά οφείλει να συμβάλει και στη διαμόρφωσή τους. Η εργασία αυτή κάθε άλλο παρά εύκολη είναι και απαιτεί ευστροφία, δημιουργικότητα και εμπειρία. Μια τακτική που συνήθως αποδίδει είναι να προκαλείται ο πελάτης με συνεχείς αμφισβητήσεις των όσων περιγράφει, έτσι ώστε να τα θέτει ο ίδιος σε αναθεώρηση από πολλές πλευρές. Η κατάληξη μιας τέτοιας διαδικασίας, ακόμα και αν δεν απέχει πολύ από το σημείο εκκίνησης, θα είναι σαφώς πιο τεκμηριωμένη.

Βασική απαίτηση είναι ο κατασκευαστής να διαθέτει σημαντικές ικανότητες αντίληψης και πνεύμα διαλόγου. Εκ των ων ουκ άνευ είναι να μπορεί ο κατασκευαστής να μπαίνει στην ουσία του προβλήματος που διαπραγματεύεται η υπό κατασκευή εφαρμογή λογισμικού, ώστε να μπορεί να ανταπεξέλθει στις απαιτήσεις του διαλόγου που αναφέραμε. Εδώ βρίσκεται και η πρόκληση που αντιμετωπίζει ο μηχανικός λογισμικού στην εργασία του, η οποία κάθε άλλο παρά μονότονη και επαναλαμβανόμενη μπορεί να χαρακτηριστεί.

#### **4.5.2. Προβλήματα προτύπων**

Τα μοντέλα παράστασης λογισμικού τα οποία παρουσιάστηκαν δεν είναι τα μοναδικά μέσα που χρησιμοποιούνται για την καταγραφή των απαιτήσεων από το λογισμικό. Η επάρκειά τους για το σκοπό αυτό έχει αρκετές φορές αμφισβητηθεί, χωρίς ωστόσο να έχει προταθεί μια καλύτερη λύση, παρά τη σημαντική ερευνητική εργασία που στρέφεται στην κατεύθυνση αυτή.

Ένας κατασκευαστής λογισμικού που ακολουθεί όσα περιγράφονται στο παρόν κεφάλαιο ολοκληρώνει την προδιαγραφή των απαιτήσεων από το λογισμικό έχοντας συμπληρώσει το έγγραφο προδιαγραφών των απαιτήσεων από το λογισμικό, έχοντας κατασκευάσει τα διαγράμματα ροής δεδομένων, οντοτήτων – συσχετίσεων και μετάβασης καταστάσεων, καθώς και έχοντας δημιουργήσει ένα λεξικό δεδομένων.

Στην ιδανική περίπτωση, το σύνολο αυτό της τεκμηρίωσης είναι επαρκές, αφενός, για να καταλάβουν όλοι οι εμπλεκόμενοι (πελάτης, κατασκευαστής) τι κάνει το λογισμικό και, αφετέρου, για να μπορέσει ο κατασκευαστής να προχωρήσει στην επόμενη φάση, αυτή της σχεδίασης. Συχνά τα παραπάνω



δεν ισχύουν, με αποτέλεσμα οι κατασκευαστές να αναγκάζονται είτε να εισάγουν νέα μέσα (διαγράμματα, κείμενα κ.λπ.) για την καταγραφή των απαιτήσεων από το λογισμικό είτε να τροποποιήσουν κατά το δοκούν τα υπάρχοντα. Η αρχική εντύπωση είναι ότι κάτι τέτοιο δεν είναι κακό εφόσον κρίνεται αναγκαίο. Ωστόσο, ο πλουραλισμός μη πρωτοτυποποιημένων διαγραμμάτων, κειμένων κ.λπ. δημιουργεί τελικά σύγχυση στον ίδιο τον κατασκευαστή που τα εισήγαγε, αφού συνήθως αυτά δεν καλύπτουν παρά πρόσκαιρες ανάγκες και δεν μελετώνται αρκετά πριν από τη χρήση τους. Το χειρότερο είναι ότι η πρακτική αυτή περιορίζει τη δυνατότητα επικοινωνίας με άλλους κατασκευαστές και εμπλεκόμενους στην ανάπτυξη του λογισμικού γενικότερα.

Δυστυχώς, οι ερευνητές που εργάζονται στη γνωστική περιοχή του λογισμικού συχνά εντείνουν τη σύγχυση εισάγοντας νέους συμβολισμούς και έννοιες με παρεμφερή σημασία, με αποτέλεσμα να δημιουργείται σύγχυση γύρω από έννοιες, σύμβολα, διαγράμματα και έγγραφα που σχετίζονται με την περιγραφή των απαιτήσεων από το λογισμικό (και όχι μόνο). Ανατρέχοντας στη βιβλιογραφία, ο αναγνώστης μπορεί να βρει αρκετές επεκτάσεις και εναλλακτικούς συμβολισμούς για όλα τα διαγράμματα που παρουσιάστηκαν στο κεφάλαιο αυτό. Κάποιες από αυτές συνδέονται με συγκεκριμένες μεθοδολογίες ανάπτυξης ή/και με συγκεκριμένα εργαλεία (CASE). Σχετικά πρόσφατα άρχισε να γνωρίζει αποδοχή το πρότυπο UML (Unified Modeling Language), το οποίο δημιουργήθηκε έχοντας κατά νου την αντικειμενοστρεφή (object-oriented) ανάπτυξη λογισμικού.

#### 4.5.3. Προβλήματα γλώσσας

Εν τη απουσία καθολικά αποδεκτών και κατανοητών διαγραμμάτων και συμβολισμών, η φυσική γλώσσα παραμένει το ισχυρότερο εργαλείο που διαθέτουμε για την περιγραφή των απαιτήσεων από το λογισμικό. Τελικά, είναι η ευστοχία και η σαφήνεια της γλώσσας που θα καθορίσει την ακρίβεια και τη σαφήνεια της περιγραφής των απαιτήσεων είτε αυτή χρησιμοποιείται ως τίτλος σε μετασχηματισμό, ροή δεδομένων, οντότητα κ.λπ. είτε ως συνταγμένο κείμενο κάποιας παραγράφου του εγγράφου προδιαγραφών των απαιτήσεων από το λογισμικό.

Η γλώσσα, ωστόσο, δεν είναι και δεν οφείλει να είναι μονοσήμαντα αντιληπτή. Συχνά περιέχει ασάφειες και διφορούμενα ή είναι ελλιπής παρά την προσπάθεια που έχει γίνει για το αντίθετο. Οι περιγραφές απαιτήσεων σε φυσική γλώσσα χρήζουν συχνά διευκρινήσεων που γίνονται επίσης με χρήση κειμένου, το οποίο μπορεί εκ νέου να περιέχει διφορούμενα και ασάφειες και το πρόβλημα να εξακολουθεί να υφίσταται. Ο σωστός χειρισμός της γλώσσας, λοιπόν, είναι απαραίτητη προϋπόθεση για την περιγραφή των απαιτήσεων από το λογισμικό. Είναι εύκολο να το διαπιστώνει κανείς, αλλά δύσκολο να το εφαρμόζει. Από γλωσσικής πλευράς, ένα τέτοιο κείμενο πρέπει να είναι λιτό, να περιέχει μικρές προτάσεις, να μην αφήνει περιθώρια παρερμηνειών ή κενά και να λέει με τρόπο εύστοχο αυτό που θέλει να πει και μόνο αυτό.

Ένα επιπλέον πρόβλημα είναι το πλήθος των όρων που προέρχονται από την αγγλική και πρέπει να αποδοθούν στην ελληνική γλώσσα. Οι αποδόσεις κάθε όρου είναι αρκετές, όχι πάντα εύστοχες (συχνά εντελώς λανθασμένες), περιέχουν διφορούμενα, γίνονται με διαφορετικό τρόπο αντιληπτές ή συγχέονται με άλλους όρους. Η χρήση των όρων στην αγγλική δεν είναι λύση, όπως δεν είναι λύση η βεβιασμένη μετάφραση των πάντων και η μετάφραση επιμονή σε μία άποψη –στη «δική μας». Οι διαμάχες μεταξύ οργανισμών, φορέων, συλλόγων κ.λπ. περί αρμοδιότητας και η επιχειρούμενη, κυρίως από περιοδικά του χώρου, *de facto* επιβολή ορολογίας απλώς εντείνουν τη σύγχυση και φθείρουν την ελληνική γλώσσα.

## Άσκηση 6/Κεφάλαιο 4

**Στο παράδειγμα της εφαρμογής λογισμικού «Επίκουρος», ο όρος «εγγραφή» αποτελεί περίπτωση όρου ο οποίος συναντάται με δύο διαφορετικές έννοιες. Εξηγήστε τις έννοιες αυτές. Μπορεί να αρθεί το διφορούμενο;**

### 4.5.4. Προβλήματα οικονομικά

Αν και ο τίτλος φαίνεται ξένος με το αντικείμενο του παρόντος κεφαλαίου, στην προδιαγραφή του λογισμικού υπάρχουν προβλήματα τα οποία ανήκουν ή μπορούν να αναχθούν στην οικονομική σφαίρα. Το λογισμικό αποτελεί προϊόν το οποίο κατασκευάζεται με σκοπό το κέρδος. Η μη χειροπιαστή υπόστασή του συχνά δεν επιτρέπει την τεκμηρίωση του κόστους του προς τον πελάτη, ο οποίος δεν αντιλαμβάνεται πάντα τι είναι αυτό για το οποίο πληρώνει και επιδιώκει να «φορτώσει» το λογισμικό με χαρακτηριστικά και λειτουργίες ελαχιστοποιώντας το κόστος. Το ακριβώς αντίθετο ισχύει για την πλευρά του κατασκευαστή, ο οποίος προσπαθεί να κάνει τα λιγότερα δυνατά με όσο το δυνατόν μεγαλύτερη αμοιβή.

Ακόμη και αν εξαιρέσουμε τις ακρότητες και από τις δύο πλευρές, γίνεται αντιληπτό ότι είναι σε αμοιβαίο όφελος να συμφωνηθεί και, μάλιστα, γραπτά η περιγραφή του υπό κατασκευή λογισμικού. Το γραπτό πάνω στο οποίο βασίζεται η συμφωνία δεν είναι άλλο από τις προδιαγραφές των απαιτήσεων από το λογισμικό ή μια περίληψη αυτών. Η συμφωνία είναι τόσο σταθερή, όσο σαφής και συγκεκριμένη είναι η προδιαγραφή των απαιτήσεων από το λογισμικό. Πολλά έργα λογισμικού έχουν καταλήξει στα δικαστήρια, με τα δύο μέρη να αποδίδουν διαφορετική ερμηνεία στα ίδια λεγόμενα, έχοντας ευσταθή επιχειρήματα, συγκρουόμενα όμως συμφέροντα. Φαίνεται, λοιπόν, ότι πέραν της τεχνικής πλευράς, η προδιαγραφή των απαιτήσεων από το λογισμικό έχει και άλλη μία, την επιχειρηματική – οικονομική.

## Δραστηριότητα 10/Κεφάλαιο 4

Στο χρηματιστήριο διαπραγματεύονται μετοχές εισηγμένων εταιρειών. Σε κάθε συνεδρίαση, κάθε μετοχή πραγματοποιεί έναν αριθμό πράξεων αγοραπωλησίας συνολικής αξίας ενός ποσού (τζίρος). Κατά τις πράξεις αυτές μας ενδιαφέρει η τήρηση της πληροφορίας του τζίρου, της μέγιστης, της ελάχιστης και της μέσης τιμής διαπραγμάτευσης, καθώς και της τιμής κλεισίματος, δηλαδή της τιμής με την οποία έγινε η τελευταία πράξη για κάθε μετοχή. Από τις πληροφορίες αυτές θέλουμε να εξάγουμε στατιστικά στοιχεία, όπως μέσος τζίρος περιόδου, διαφορά τιμής περιόδου, μέγιστη και ελάχιστη τιμή περιόδου.

Με βάση την παραπάνω περιγραφή, δώστε τουλάχιστον τρεις λειτουργικές απαιτήσεις από μια εφαρμογή λογισμικού που χρησιμοποιείται για τον σκοπό αυτό. Κατασκευάστε ένα μέρος του διαγράμματος ροής δεδομένων, καθώς και το διάγραμμα οντοτήτων – συσχετίσεων που αντιστοιχεί στην εφαρμογή αυτή.

## ΕΝΟΤΗΤΑ 4.6. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ ΑΥΤΟΑΞΙΟΛΟΓΗΣΗΣ

### 4.6.1. Δραστηριότητες

#### Δραστηριότητα 1/Κεφάλαιο 4

---

**Λειτουργικές απαιτήσεις:** 1) Εκτέλεση της πράξης «πρόσθεση». 2) Εκτέλεση της πράξης «αφαίρεση». 3) Εκτέλεση της πράξης «πολλαπλασιασμός». 4) Εκτέλεση της πράξης «διαίρεση». 5) Εκτέλεση υπολογισμού ποσοστού επί τοις εκατό. 6) Μηδενισμός καταχωρητή.

**Μη λειτουργικές απαιτήσεις:** 1) Η μορφή του να θυμίζει τη γνωστή μορφή του υπολογιστή τσέπης. 2) Να χειρίζεται πραγματικούς αριθμούς 10 ψηφίων. 3) Να δέχεται εντολές από το ποντίκι και από το πληκτρολόγιο. 4) Να μπορεί να ενεργοποιείται με ένα συνδυασμό πλήκτρων μέσα από οποιοδήποτε πρόγραμμα.

Όπως φαίνεται, οι λειτουργικές απαιτήσεις περιγράφουν κάτι που θέλουμε να κάνει το λογισμικό, ενώ οι μη λειτουργικές περιγράφουν κάποια χαρακτηριστικά που θέλουμε να έχει. Μπορείτε να ορίσετε πολύ περισσότερες απαιτήσεις ρίχνοντας μια προσεκτική ματιά σε ένα οποιοδήποτε πρόγραμμα τύπου calculator, όπως αυτό των Windows 98. Για παράδειγμα, μπορείτε να διαπιστώσετε τη δυνατότητα εμφάνισης ως απλού ή επιστημονικού υπολογιστή τσέπης και το πλήθος των πράξεων που διατίθενται στη δεύτερη περίπτωση.

#### Δραστηριότητα 2/Κεφάλαιο 4

---

**Απαιτήσεις από το σύστημα:** 1) Λήψη και αποστολή μέτρησης θερμοκρασίας. 2) Λήψη και αποστολή μέτρησης υγρασίας. 3) Λήψη και αποστολή μέτρησης πίεσης.

**Απαιτήσεις από το λογισμικό:** 1) Αποθήκευση μετρήσεων θερμοκρασίας. 2) Αποθήκευση μετρήσεων πίεσης. 3) Αποθήκευση μετρήσεων υγρασίας. 4-6) Επεξεργασία μετρήσεων (θερμοκρασίας, πίεσης,

υγρασίας) και υπολογισμός μέσης τιμής, μέγιστης – ελάχιστης τιμής και τυπικής απόκλισης.

**Κατανομή στις συνιστώσες του συστήματος:** Μηχανές: Λήψη, αποστολή και αποθήκευση μετρήσεων. Λογισμικό: επεξεργασία μετρήσεων. Άνθρωποι: Επίβλεψη λειτουργίας και αξιολόγηση μετρήσεων.

Μπορείτε να εμπλουτίσετε την περιγραφή αν ως συνιστώσα του συστήματος θεωρήσετε και το τηλεπικοινωνιακό δίκτυο, το οποίο μπορεί να είναι ενσύρματο ή ασύρματο, ανταποκρίνεται με συγκεκριμένες απαιτήσεις επιδόσεων κ.ά. Ο διαχωρισμός των απαιτήσεων από το σύστημα και το λογισμικό δεν είναι πάντα εύκολη υπόθεση. Συχνά μπορεί κανείς να επιχειρηματολογήσει για τον χαρακτηρισμό μιας απαίτησης και στις δύο κατηγορίες. Αν αυτό σας δημιουργεί σύγχυση, μπορείτε να ξαναδιαβάσετε την Ενότητα 4.1.

Ένας άλλος πρακτικός τρόπος αντιμετώπισης είναι ο εξής: χαρακτηρίστε προσωρινά την υπό αμφισβήτηση απαίτηση ως απαίτηση από το λογισμικό και αφήστε σε εκκρεμότητα τον οριστικό χαρακτηρισμό της. Σε επόμενο βήμα της ανάπτυξης, και ιδιαίτερα στη δημιουργία του διαγράμματος ροής δεδομένων, θα πρέπει να είστε σε θέση να διαπιστώσετε αν πράξατε ορθά: αν βρείτε μετασχηματισμό που να αντιστοιχεί στην υπό αμφισβήτηση απαίτηση, τότε πράγματι πρόκειται για απαίτηση από το λογισμικό. Διαφορετικά, πρόκειται για απαίτηση από το σύστημα.

## Δραστηριότητα 3/Κεφάλαιο 4

Η ανάλυση απαιτήσεων γίνεται σε στενή επαφή με τον πελάτη και στοχεύει στην κατανόηση του προβλήματος από τον κατασκευαστή. Αποτέλεσμα αυτής της εργασίας είναι μια λίστα απαιτήσεων και τρία διαγράμματα, τα οποία είναι κατανοητά από τον πελάτη και χρήσιμα στον κατασκευαστή για τη συνέχιση της εργασίας του.

Η διάκριση και προδιαγραφή κάθε συγκεκριμένης απαίτησης γίνεται από τον μηχανικό λογισμικού και περιγράφει με σημαντικά μεγαλύτερη λεπτομέρεια και αυστηρή δομή κάθε απαίτηση από το λογισμικό, με αποτέλεσμα τη σύνταξη του εγγράφου «προδιαγραφές των απαιτήσεων από το λογισμικό».



Με απλά λόγια, σκεφτείτε το πρώτο σαν ένα σύνολο συνεντεύξεων και (κατά τη γνώμη μας) ιδιαίτερα δημιουργικών συζητήσεων, ενώ το δεύτερο σαν το προϊόν μιας αναλυτικής επεξεργασίας των αποτελεσμάτων των συνεντεύξεων. Αν συλλάβατε την διαφορά, μπορείτε με εμπιστοσύνη να επιχειρήσετε τη σύνταξη ενός εγγράφου προδιαγραφών των απαιτήσεων από το λογισμικό. Αν όχι, μην ανησυχείτε. Η σύνταξη του εγγράφου προδιαγραφών των απαιτήσεων από το λογισμικό είναι η διαδικασία που επιδέχεται τις περισσότερες ερμηνείες στην ανάπτυξη του λογισμικού. Διαβάστε ξανά τις Ενότητες 4.2 και 4.3 και συνεχίστε επαληθεύοντας σε κάθε βήμα τα παραγόμενα προϊόντα σας.

## Δραστηριότητα 4/Κεφάλαιο 4

---

Μπορεί κανείς να εντοπίσει αρκετά αλληλοσυγκρουόμενα χαρακτηριστικά ενός τέτοιου εγγράφου, μιας και θέλουμε να είναι σύντομο, πλήρες και χρήσιμο. Αυτά τα τρία χαρακτηριστικά είναι αλληλοσυγκρουόμενα ανά δύο μεταξύ τους. Δύο σχετικές διατυπώσεις είναι οι ακόλουθες:

1. Ένα έγγραφο που είναι αρκετά λεπτομερές ώστε να είναι χρήσιμο στη συντήρηση του λογισμικού δεν είναι πάντα εύκολο να αλλαχτεί καθώς για την ενσωμάτωση αλλαγών απαιτείται «βαθύ σκάψιμο».
2. Ένα έγγραφο που περιγράφει τη συμπεριφορά του λογισμικού σε ανεπιθύμητες καταστάσεις μπορεί να χρειάζεται να αναφερθεί σε εσωτερικά χαρακτηριστικά του λογισμικού, πράγμα ανεπιθύμητο στη φάση αυτή.

Δεν είναι απαραίτητο τα παραπάνω να είναι αντιληπτά από τον αναγνώστη με την πρώτη. Βέβαια, αξίζει ένα μπράβο σε αυτόν που έδωσε ακόμη περισσότερες από τις απαντήσεις που παρατέθηκαν εδώ. Η εμπειρία στη σύνταξη τέτοιων εγγράφων είναι που αποδίδει την ωριμότητα που χρειάζεται για να διατηρούνται οι πολύ λεπτές ισορροπίες μεταξύ των «επιθυμητών χαρακτηριστικών» του εγγράφου προδιαγραφών των απαιτήσεων από το λογισμικό. Τα πρότυπα είναι καλά, αλλά δεν είναι πάντα εύκολο να ακολουθηθούν.

### 3.2.6 Λειτουργική απαίτηση A6

**Καταχώρηση βαθμολογίας:** Η εφαρμογή εμφανίζει φόρμα μέσω της οποίας ο χρήστης ενημερώνει, δίνει στοιχεία σπουδαστή, μαθήματος και βαθμολογίας, το αρχείο της οποίας ενημερώνεται. **Είσοδος:** Στοιχεία σπουδαστή, μαθήματος, βαθμός. **Επεξεργασία:** Έλεγχος αν έχει γίνει εγγραφή του σπουδαστή στο μάθημα και, αν ναι, τότε ετοιμασία και επαλήθευση της εγγραφής. **Έξοδοι:** Ενημερωμένο αρχείο βαθμολογίας ή μήνυμα λάθους.

### 3.2.7 Λειτουργική απαίτηση A7

**Εκτύπωση αρχείου σπουδαστών:** Η εφαρμογή τυπώνει στον εκτυπωτή ολόκληρο το αρχείο σπουδαστών. **Είσοδος:** Αρχείο σπουδαστών. **Επεξεργασία:** Μορφοποίηση εκτύπωσης. **Έξοδοι:** Εκτύπωση στον εκτυπωτή.

### 3.2.8 Λειτουργική απαίτηση A8

**Εκτύπωση βαθμολογίας σπουδαστή:** Η εφαρμογή εμφανίζει διάλογο ερώτησης των στοιχείων του σπουδαστή και ετοιμάζει την εκτύπωση της βαθμολογίας αυτού σε όλα τα μαθήματα, την οποία στέλνει στον εκτυπωτή. **Είσοδος:** Στοιχεία σπουδαστή. **Επεξεργασία:** Έλεγχος ύπαρξης σπουδαστή, έλεγχος ύπαρξης εγγραφών βαθμολογίας, ετοιμασία και μορφοποίηση της εκτύπωσης. **Έξοδοι:** Η εκτύπωση στον εκτυπωτή του συστήματος ή μήνυμα λάθους.

Ασφαλώς ο αναγνώστης μπορεί να έχει διαφορετική άποψη για τη συμπεριφορά του λογισμικού κατά την πραγματοποίηση κάποιας λειτουργίας. Αυτό είναι φυσιολογικό και αποδεκτό. Με κανένα τρόπο οι απαντήσεις που προτείνονται εδώ δεν είναι οι μοναδικές ή οι καλύτερες. Σκοπός είναι η ανάδειξη του τρόπου και της σημασίας της περιγραφής των απαιτήσεων και δευτερευόντως η βέλτιστη λύση. Ο αναγνώστης ενθαρρύνεται να δώσει και δικές του λύσεις.



## Δραστηριότητα 6/Κεφάλαιο 4

---

Για καθεμία από τις απαιτήσεις που παρατίθενται αρχικά στη λίστα, αναζητήστε ένα μετασχηματισμό στα διαγράμματα ροής δεδομένων που να αντιστοιχεί σε αυτή. Αν βρείτε, τότε εντοπίσατε τον συσχετισμό. Αν όχι, τότε ενδεχομένως να υπάρχουν δύο ή και περισσότερες απαιτήσεις της λίστας που αντιστοιχούν στον ίδιο μετασχηματισμό.

Στην πράξη είναι πολύ χρήσιμη η αντιστοίχιση αυτή. Μπορούν να εντοπιστούν είτε νέες απαιτήσεις (δηλαδή μετασχηματισμοί που καταγράφηκαν στο διάγραμμα ροής δεδομένων, αλλά δεν περιέχονται ως απαιτήσεις αρχικά) είτε απαιτήσεις που πρέπει να καταργηθούν ή να επαναδιατυπωθούν (δηλαδή απαιτήσεις που δεν αντιστοιχούν σε κανέναν από τους μετασχηματισμούς του διαγράμματος ροής δεδομένων).

## Δραστηριότητα 7/Κεφάλαιο 4

---

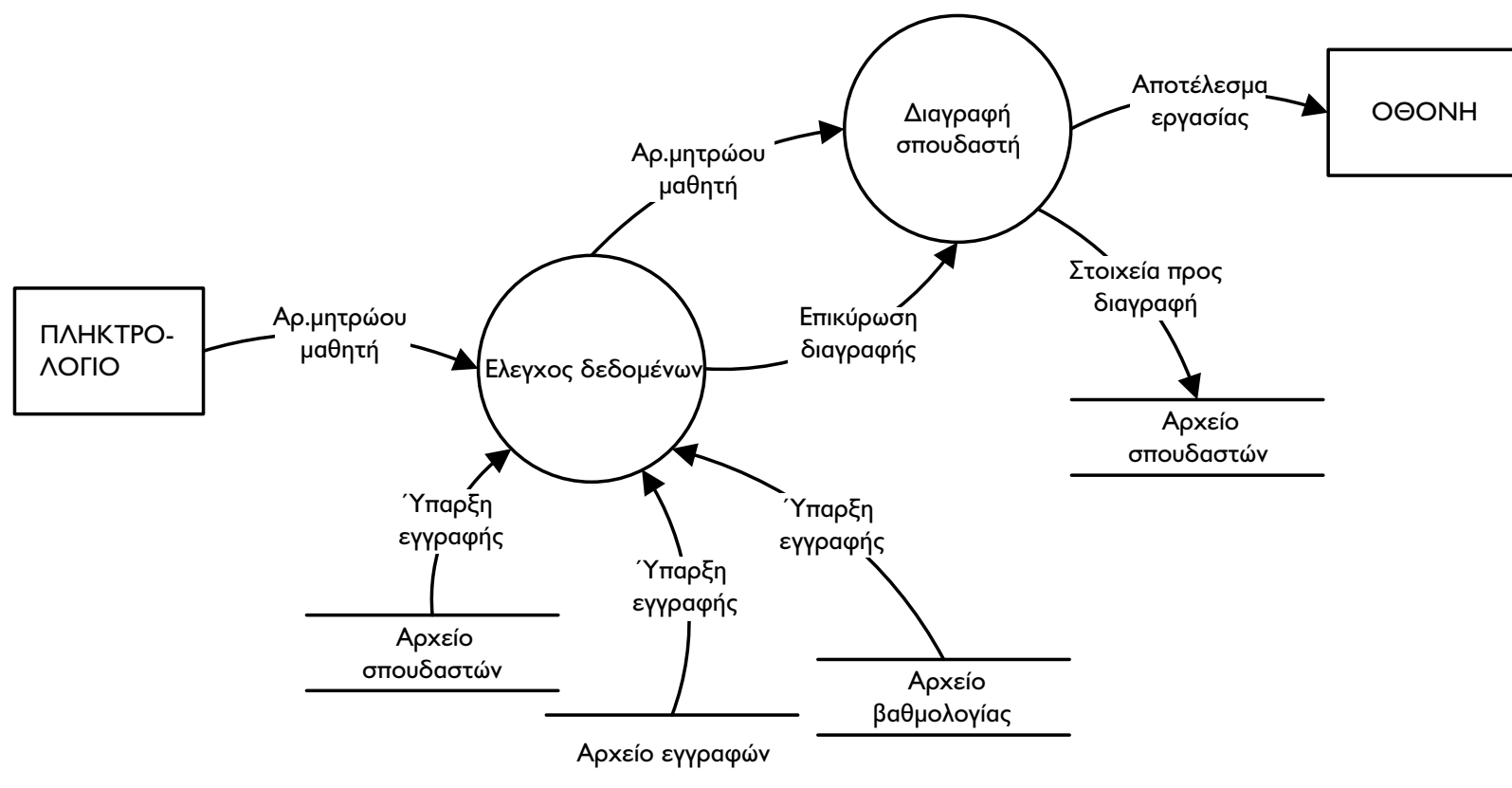
Θεωρήστε τις ροές εισόδου σε κάθε μετασχηματισμό στο επίπεδο με τη μικρότερη λεπτομέρεια και αντιστοιχίστε τις σε ροές στους αντίστοιχους μετασχηματισμούς στο επίπεδο με τη μεγαλύτερη λεπτομέρεια. Θα διαπιστώσετε σημασιολογική συνάφεια. Για παράδειγμα, η ροή «στοιχεία μαθητών» αντιστοιχεί στις ροές «στοιχεία μαθητή», «στοιχεία καθηγητή» και «στοιχεία μαθήματος».

## Δραστηριότητα 8/Κεφάλαιο 4

---

Ένα προτεινόμενο διάγραμμα ροής δεδομένων για τη ζητούμενη εργασία είναι το ακόλουθο:

Σχήμα 4.22



Ο μετασχηματισμός «έλεγχος δεδομένων» χρησιμοποιείται για να ελέγξει αν ο «αριθμός μητρώου σπουδαστή» υπάρχει στις αποθήκες δεδομένων. Στην περίπτωση αυτή δίνει τιμή στο δεδομένο «επίτρεψη διαγραφής» το οποίο μαζί με τον «αριθμό μητρώου σπουδαστή» τροφοδοτεί το μετασχηματισμό «διαγραφή σπουδαστή», ο οποίος και πραγματοποιεί την εργασία και επιστρέφει το αποτέλεσμα στο χρήστη.

## Δραστηριότητα 9/Κεφάλαιο 4

---

Τα πράγματα είναι πολύ εύκολα. Από την περιγραφή φαίνεται ότι ο μοναδικός λόγος εξόδου από την κατάσταση «σε τρέχουσα διδασκαλία» είναι το γεγονός «αρχειοθέτηση». Τα υπόλοιπα γεγονότα διατηρούν την κατάσταση «σε τρέχουσα διδασκαλία». Η απάντηση που θα δώσετε θα πρέπει να ομοιάζει αρκετά στο Σχήμα 4.21.

Αν δεν τα καταφέρετε με την πρώτη, μην απογοητεύεστε. Τα διαγράμματα καταστάσεων είναι ίσως τα πιο δυσνόητα από τα μοντέλα παράστασης λογισμικού που περιγράψαμε. Διαβάστε πάλι την Ενότητα 4.4.4 και ξαναπροσπαθήστε.

## Δραστηριότητα 10/Κεφάλαιο 4

---

Όπως αντιλαμβάνεται ο αναγνώστης, πρόκειται για ιδιαίτερα επίκαιρο παράδειγμα, καταλλήλως περιορισμένο στους εκπαιδευτικούς σκοπούς του. Ακολουθώς δίνεται μια πρώτη προσέγγιση. Ο αναγνώστης ενθαρρύνεται να τη συμπληρώσει και, γιατί όχι, να ολοκληρώσει την εφαρμογή που παρουσιάζεται εδώ.

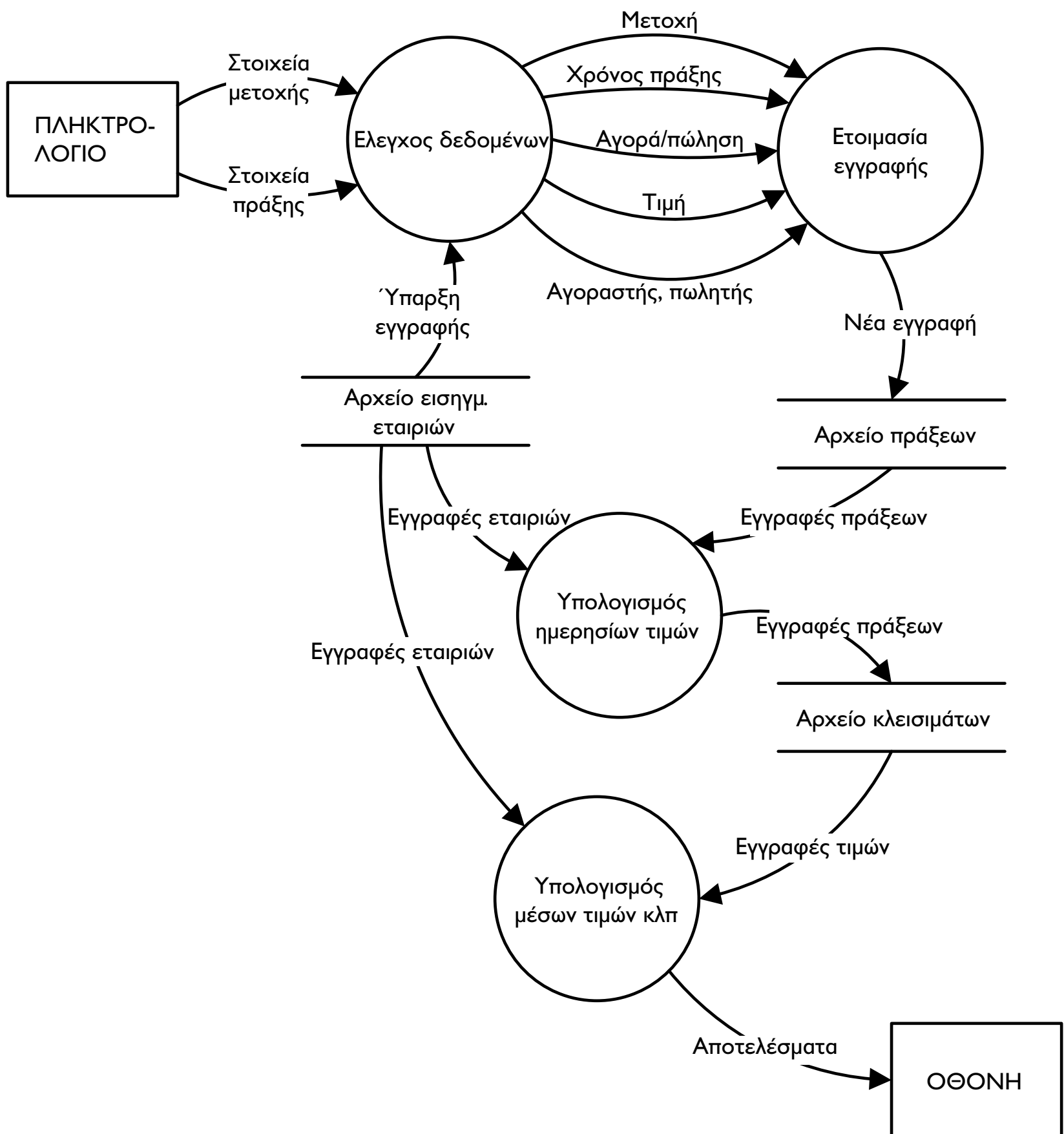
Λειτουργικές απαιτήσεις:

1. Τήρηση αρχείου εισηγμένων εταιρειών.
2. Τήρηση αναλυτικού ημερολογίου χρηματιστηριακών πράξεων.
3. Τήρηση αρχείου κλεισιμάτων.
4. Υπολογισμός ημερησίου μέγιστου, ελάχιστου, μέσης τιμής, τιμής κλεισίματος, όγκου συναλλαγών.
5. Υπολογισμός μέσου τζίρου περιόδου.

6. Υπολογισμός μέγιστης και ελάχιστης τιμής περιόδου.
7. Υπολογισμός μεταβολής τιμής περιόδου.

## Διάγραμμα ροής δεδομένων:

Σχήμα 4.23



**Παρατηρήσεις:** Ο «υπολογισμός μέσω τιμών» κ.λπ. μπορεί να αναλυθεί σε ένα ακόμη επίπεδο λεπτομέρειας, έτσι ώστε να εντοπιστούν οι τέσσερις μετασχηματισμοί που αντιστοιχούν στις λειτουργικές απαιτήσεις 4-7. Το αρχείο πράξεων περιέχει ένα τεράστιο πλήθος εγγραφών, ενώ το αρχείο κλεισιμάτων μόνο μία εγγραφή την ημέρα για κάθε εισηγμένη εταιρεία.

## Διάγραμμα οντοτήτων – συσχετίσεων:

Σχήμα 4.23



## 4.6.2. Ασκήσεις αυτοαξιολόγησης

### Άσκηση 1/Κεφάλαιο 4

---

**Λειτουργικές απαιτήσεις: K2-KI4. Μη λειτουργικές απαιτήσεις: KI.**

Οφείλουμε να σας συγχαρούμε αν βρήκατε την απάντηση με την πρώτη. Αν η πρώτη σας άποψη είναι διαφορετική, μπορείτε να ανατρέξετε και πάλι στην Ενότητα 4.1.3 και να ξαναπροσπαθήσετε.

### Άσκηση 2/Κεφάλαιο 4

---

1) Μη λειτουργική, φυσική. 2) Λειτουργική. 3) Μη λειτουργική, φυσική. 4) Μη λειτουργική, σχεδίασης. 5) Λειτουργική. 6) Μη λειτουργική, χρήσης. 7) Μη λειτουργική, υλοποίησης. 8) Μη λειτουργική, επικοινωνίας με άλλα συστήματα. 9) Μη λειτουργική, αξιοπιστίας. 10) Λειτουργική.

Ο χαρακτηρισμός των απαιτήσεων, και ιδιαίτερα η ταξινόμηση των μη λειτουργικών απαιτήσεων, μπορεί να είναι μια ιδιαίτερα δύσκολη εργασία, ιδίως όταν η διατύπωση επιδέχεται περισσότερες από μία ερμηνείες. Προσπαθήστε να επιβεβαιώσετε τις απαντήσεις που δίνονται εδώ ανατρέχοντας στην ύλη της Ενότητας 4.1.3 και στο Σχήμα 4.2.



### Άσκηση 3/Κεφάλαιο 4

---

Κατά την ανάλυση απαιτήσεων κατασκευάζονται τρία διαγράμματα, ενώ η διάκριση και προδιαγραφή κάθε συγκεκριμένης απαίτησης γίνεται από τον κατασκευαστή με σκοπό τη συγγραφή του σχετικού εγγράφου. Πρόκειται δηλαδή για διαφορετικές εργασίες, η πρώτη από τις οποίες δίνει μια γενική περιγραφή της εφαρμογής με τη βοήθεια διαγραμμάτων, ενώ η δεύτερη μπαίνει σε μεγαλύτερο βάθος και αναλύει κάθε συγκεκριμένη απαίτηση διεξοδικότερα.

Είναι φυσιολογικό να μην μπορεί κανείς να εντοπίσει τη διάκριση αυτή με την πρώτη φορά. Συχνά, εξάλλου, στην πράξη, οι εργασίες αυτές είναι συμπληρωματικές και συμβαίνουν παράλληλα. Μπορείτε να ανατρέξετε στα γραφόμενα στην Ενότητα 4.2 και ιδιαίτερα στο Σχήμα 4.3.

### Άσκηση 4/Κεφάλαιο 4

---

Στα διαγράμματα ροής δεδομένων εμφανίζονται δύο αρχεία, το αρχείο εγγραφών και βαθμολογίας αντίστοιχα. Αυτά δεν εμφανίζονται στο διάγραμμα οντοτήτων – συσχετίσεων. Ο λόγος είναι ότι στην πραγματικότητα τα δύο αυτά αρχεία παριστάνουν ιδιώματα των δύο σχέσεων M:N που υπάρχουν μεταξύ των οντοτήτων «σπουδαστής» και «μάθημα» και πρόκειται να «αποκαλυφθούν» στη φάση της σχεδίασης. Το γεγονός αυτό δεν αποτελεί έλλειψη στη σχεδίαση των συγκεκριμένων διαγραμμάτων ροής δεδομένων, αλλά πρόκειται για αδυναμία του διαγράμματος ροής δεδομένων να περιλάβει τέτοιες πληροφορίες, διότι κάτι τέτοιο είναι εκτός των στόχων του.

Με την ευκαιρία, στο σημείο αυτό τονίζεται η αναγκαία συμπληρωματικότητα των μοντέλων παράστασης λογισμικού, η οποία αναφέρθηκε.

### Άσκηση 5/Κεφάλαιο 4

---

Ευθέως:

- Κάθε φοιτητής εγγράφεται σε περισσότερα του ενός μαθήματα.
- Κάθε φοιτητής βαθμολογείται σε περισσότερα του ενός μαθήματα.

Αντίστροφα:

- Σε κάθε μάθημα εγγράφονται πολλοί φοιτητές.
- Σε κάθε μάθημα λαμβάνουν βαθμολογία πολλοί φοιτητές.

## Άσκηση 6/Κεφάλαιο 4

---

Ο όρος «εγγραφή» χρησιμοποιείται σε δύο περιπτώσεις:

- Με την έννοια που έχει στις βάσεις δεδομένων για να περιγράψει την καταγραφή μιας οντότητας σε έναν πίνακα.
- Με την έννοια της **εγγραφής σε μάθημα**, η οποία σχετίζεται καθαρά με το πεδίο του προβλήματος που εξετάζουμε.

Το διφορούμενο δεν μπορεί παρά να επιλύεται από τα συμφραζόμενα. Δηλαδή, ανάλογα με το νόημα της έκφρασης στην οποία γίνεται η αναφορά στον όρο, γίνεται αντιληπτό και το νόημα αυτού. Ένας άλλος τρόπος είναι να καταφεύγουμε στη χρήση όρων από την αγγλική, πράγμα που πιστεύουμε ότι θα πρέπει να γίνεται μόνο όταν είναι *απολύτως* αναγκαίο.

Γενικά, χρειάζεται μεγάλη προσοχή στη σωστή χρήση των όρων. Ίσως φαίνεται κουραστικό αλλά είναι συνετό να θεωρούμε ότι δεν εννοείται τίποτε απολύτως, όλα πρέπει να γράφονται με τρόπο ώστε να επιδέχονται όσο το δυνατόν λιγότερες ερμηνείες. Αυτό δεν είναι καθόλου εύκολο, αποτελεί όμως ενδιαφέρουσα πρόκληση. Η Τεχνολογία Λογισμικού είναι από τους λίγους κλάδους της μηχανικής όπου ο επαρκής χειρισμός του λόγου είναι άκρως απαραίτητος.

## ΒΙΒΛΙΟΓΡΑΦΙΑ

Boehm, B. W. (1975) In Horowitz, E. et al., *Practical Strategies for Developing Large Software Systems*, Reading, Massachusetts: Addison-Wesley.

Boehm, B. W., *Verifying and Validating Software Requirements and Design Specifications*, IEEE Software, January 1984, pp.75-88, 1984.

De Macro, T., *Structured Analysis and System Specification*, Yourdon Press, New York, 1978.

Gane, C. and T. Sarson *Structured Systems Analysis: Tools and Techniques*, Prentice-Hall, Englewood Cliffs, N. J. 1979.

Hall, A., *Seven Myths of Formal Methods*, IEEE Software, September 1990, pp. 11-19, 1990.

*IEEE Guide To Software Requirements Specification*, ANSI/IEEE, Std 830-1993, 1984.

Pressman, R. S., *Software Engineering-A Practitioners Approach*, McGraw-Hill.

Sommerville, I. *Software Engineering*, London: Addison-Wesley.

Yourdon, E., *Modern Structured Analysis*, Yourdon Press, Prentice Hall Building, Englewood Cliffs, New Jersey 07632, 1989.

## ΔΟΜΗΜΕΝΗ ΣΧΕΔΙΑΣΗ

Σκοπός του κεφαλαίου είναι η παρουσίαση μεθοδολογιών κατάστρωσης και τρόπων περιγραφής του σχεδίου του λογισμικού, σύμφωνα με την προσέγγιση της δομημένης σχεδίασης. Η εμβέλεια αναφοράς στη σχεδίαση καλύπτει τα σχέδια διάταξης, την αρχιτεκτονική σχεδίαση, τη σχεδίαση διεπαφών, τη λεπτομερή σχεδίαση μονάδων, καθώς και –επιγραμματικά– τη σχεδίαση σχεσιακών βάσεων δεδομένων.

Μετά τη μελέτη του κεφαλαίου αυτού, ο αναγνώστης θα είναι σε θέση:

- να ορίζει είναι σχέδιο λογισμικού και να αναφέρει τέσσερα κριτήρια ποιότητας αυτού,
- να αναφέρει τέσσερις επιμέρους εργασίες που εκτελούνται κατά τη σχεδίαση λογισμικού και τα προϊόντα αυτών,
- να ορίζει την έννοια της διάταξης λογισμικού και να αναφέρει τέσσερις τέτοιες διατάξεις,
- να χρησιμοποιεί τεχνικές δομημένης σχεδίασης, ώστε να εντοπίζει τις έννοιες του κεντρικού μετασχηματισμού και του κέντρου δοσοληψιών σε διαγράμματα ροής δεδομένων,
- να κατασκευάζει σχέδια δομής προγράμματος ξεκινώντας από διαγράμματα ροής δεδομένων και εφαρμόζοντας τέσσερα βήματα,
- να κατασκευάζει λεπτομερή σχέδια μονάδων λογισμικού χρησιμοποιώντας περιγραφή με ψευδοκώδικα,
- να περιγράφει τις εργασίες και τις αρχές που πρέπει να ακολουθούνται κατά τη σχεδίαση δεδομένων.

- Δομημένη σχεδίαση
- Κεντρικός μετασχηματισμός
- Κέντρο δοσοληψιών
- Διάγραμμα δομής προγράμματος
- Έγγραφο περιγραφής του σχεδίου του λογισμικού
- Γλώσσα περιγραφής προγράμματος
- Ψευδοκώδικας

## Σύνοψη

---

Κατά τη σχεδίαση λογισμικού προσδιορίζεται το πώς θα εκτελούνται οι επιθυμητές από το λογισμικό εργασίες οι οποίες έχουν καθοριστεί κατά την προδιαγραφή των απαιτήσεων από το λογισμικό. Το έγγραφο και τα διάφορα μοντέλα περιγραφής λογισμικού που έχουν κατασκευαστεί κατά την προδιαγραφή των απαιτήσεων αποτελούν την είσοδο στη σχεδίαση. Ο σχεδιαστής λογισμικού, χρησιμοποιώντας αυτά ως αρχικά δεδομένα, τις μεθοδολογίες της Τεχνολογίας Λογισμικού ως γραμμή πλεύσης και τη δημιουργικότητα και την εμπειρία του ως οδηγό, αναλαμβάνει να καθορίσει τον πιο πρόσφορο –στις εκάστοτε συνθήκες– τρόπο υλοποίησης του λογισμικού. Οι επιμέρους εργασίες που εκτελεί είναι η αρχιτεκτονική σχεδίαση, η σχεδίαση διεπαφών, η λεπτομερής σχεδίαση μονάδων και η σχεδίαση δεδομένων. Σε αρκετές περιπτώσεις, όπως στη σχεδίαση διεπαφών και στη σχεδίαση δεδομένων, καλείται να επιστρατεύσει γνώσεις από άλλες, περισσότερο εξειδικευμένες, γνωστικές περιοχές, όπως αυτή της επικοινωνίας ανθρώπου – μηχανής και των βάσεων δεδομένων. Η μεθοδολογία της δομημένης σχεδίασης που παρουσιάζεται στο κεφάλαιο αυτό προσφέρει έναν συστηματικό τρόπο για μετάβαση από τα διαγράμματα ροής δεδομένων σε διαγράμματα δομής προγράμματος. Τα αποτελέσματα της σχεδίασης περιγράφονται με τη βοήθεια του εγγράφου περιγραφής του σχεδίου του λογισμικού, διαγραμμάτων διάταξης, διαγραμμάτων δομής προγράμματος και ψευδοκώδικα.

Το πρόβλημα που αντιμετωπίζεται κατά τη φάση της προδιαγραφής των απαιτήσεων είναι το τι θα κάνει το λογισμικό, καθώς και το ποια θα είναι τα χαρακτηριστικά του ιδιώματα. Το αποτέλεσμα της φάσης αυτής είναι το έγγραφο προδιαγραφής των απαιτήσεων από το λογισμικό, καθώς και ένα σύνολο μοντέλων παράστασης λογισμικού σε μορφή διαγραμμάτων ροής δεδομένων, οντοτήτων – συσχετίσεων και μετάβασης καταστάσεων, μαζί με το λεξικό δεδομένων, τα οποία παρουσιάστηκαν αναλυτικά στο Κεφάλαιο 4. Στην επόμενη φάση, αυτή της σχεδίασης, θεωρείται γνωστό το τι θα κάνει το λογισμικό και αντιμετωπίζεται το πρόβλημα του πώς θα το κάνει. Όλα τα προϊόντα που έχουν παραχθεί κατά τη φάση προδιαγραφής των απαιτήσεων από το λογισμικό αποτελούν την είσοδο στη φάση της σχεδίασης, δηλαδή το υλικό με το οποίο ο σχεδιαστής λογισμικού θα χτίσει το κατασκευάσμα του. Η διαδικασία αυτή δεν είναι απλή ούτε και αυτοματοποιείται εύκολα. Οι προδιαγραφές συχνά είναι ασαφείς, ελλιπείς ή και αλληλοσυγκρουόμενες. Ακόμη περισσότερο, υπάρχουν πολλοί τρόποι να ικανοποιηθούν, δηλαδή περισσότερα από ένα σχέδια μπορεί να ικανοποιούν τις ίδιες προδιαγραφές. Ο σχεδιαστής μπαίνει στο στίβο αναζήτησης του καλύτερου, με το καλύτερο να ορίζεται από ένα πλήθος τεχνικά και μη κριτήρια (όπως κόστος και χρόνος υλοποίησης). Αυτό είναι που κάνει τη δουλειά του δημιουργική και ενδιαφέρουσα. Η Τεχνολογία Λογισμικού προσπαθεί να του παρέχει μεθοδολογίες, εργαλεία και κριτήρια αποτίμησης της εργασίας του.

### ΕΝΟΤΗΤΑ 5.1. ΣΚΟΠΟΣ ΤΗΣ ΣΧΕΔΙΑΣΗΣ

Στην ενότητα αυτή θα οριστεί το αντικείμενο της σχεδίασης του λογισμικού και θα οριοθετηθούν οι στόχοι τους οποίους καλείται να εκπληρώσει ο μηχανικός λογισμικού που ασχολείται με τη σχεδίαση.

Αυτό που ζητείται από τη σχεδίαση είναι ένας τρόπος περιγραφής της κατασκευής του λογισμικού, έτσι ώστε αυτό να ικανοποιεί τις προδιαγραφές που έχουν τεθεί, δηλαδή να μπορεί να εκτελεί τις επιθυμητές λειτουργίες και να έχει τα επιθυμητά χαρακτηριστικά. Η ύπαρξη των προδιαγραφών



είναι αναγκαία για να ξεκινήσει η σχεδίαση και επιβάλλεται από τις αρχές της Τεχνολογίας Λογισμικού. Η περιγραφή αυτή, που παράγεται κατά τη σχεδίαση, ονομάζεται «σχέδιο λογισμικού».

### **Σχέδιο λογισμικού:**

Σχέδιο λογισμικού είναι η περιγραφή των μονάδων που αποτελούν το λογισμικό, των συσχετίσεων μεταξύ τους, της διάταξής τους, καθώς και της εσωτερικής τους λεπτομέρειας.

Η παραγωγή του σχεδίου λογισμικού είναι αναγκαία, προκειμένου να γίνει δυνατή η κατασκευή του, όπως άλλωστε ισχύει για κάθε τεχνικό έργο. Η λύση στο πρόβλημα της σχεδίασης λογισμικού δεν είναι καθόλου εύκολη. Για κάθε προδιαγραφή δεν είναι παράλογο να θεωρούμε ότι μπορούμε να κατασκευάσουμε περισσότερα του ενός σχέδια, δηλαδή να θεωρούμε ότι μπορεί να υλοποιηθεί με περισσότερους του ενός τρόπους. Μερικές από τις σημαντικότερες πλευρές του προβλήματος της σχεδίασης είναι οι ακόλουθες:

- Με ποια στρατηγική πρέπει να αντιμετωπίσουμε τη μετάβαση από τις προδιαγραφές στη σχεδίαση, έτσι ώστε η εργασία μας να είναι αποτελεσματική;
- Ποιος από τους τρόπους που μπορούμε να σκεφτούμε για την υλοποίηση μιας προδιαγραφής είναι ο καλύτερος και πώς τεκμηριώνεται αυτό;
- Σε ποιο βαθμό δεσμεύεται η σχεδίαση από το περιβάλλον (γλώσσα προγραμματισμού, εργαλεία, λειτουργικό σύστημα) στο οποίο θα γίνει η κατασκευή του προγράμματος;
- Ποια είναι η πιο επαρκής, δηλαδή κατάλληλη, χωρίς να είναι ούτε ελλιπής ούτε φλύαρη, περιγραφή του σχεδίου του λογισμικού;
- Πώς εξασφαλίζεται η ποιότητα του παραγόμενου λογισμικού μέσα από τις εργασίες που λαμβάνουν χώρα κατά τη σχεδίαση;

Σκοπός της σχεδίασης είναι να δώσει την καλύτερη δυνατή –στις εκάστοτε συνθήκες– απάντηση στα παραπάνω ερωτήματα. Ας σημειωθεί ότι δεν

υπάρχει η καλύτερη κατ' απόλυτη έννοια λύση, παρά μόνο η καλύτερη δυνατή στις εκάστοτε συνθήκες. Η εποχή που επιδίωξη των μηχανικών λογισμικού ήταν η εύρεση της απόλυτα καλύτερης και γενικής λύσης στο πρόβλημα της σχεδίασης έχει δώσει τη θέση της στον ρεαλισμό της επιδίωξης της βέλτιστης λύσης μέσα σε κάθε συγκεκριμένο περιβάλλον κατασκευής λογισμικού.

Παρά τον υποκειμενισμό που γενικά διέπει το χαρακτηρισμό ενός σχεδίου λογισμικού, είναι χρήσιμο να συμφωνούνται κριτήρια ποιότητας στα οποία να αποδίδεται η προσήκουσα κατά περίπτωση βαρύτητα. Τέσσερα τέτοια κριτήρια είναι τα ακόλουθα:

- Το σχέδιο πρέπει να ικανοποιεί όλες τις προδιαγραφές των απαιτήσεων από το λογισμικό (λειτουργικές και μη).
- Το σχέδιο πρέπει να περιγράφει πλήρως όλες τις πλευρές του λογισμικού: δεδομένα, λειτουργίες και συμπεριφορά, όπως αυτές θεωρούνται από την πλευρά του κατασκευαστή.
- Το σχέδιο πρέπει να είναι εύκολα κατανοητό από αυτούς που θα κληθούν να συγγράψουν τον πηγαίο κώδικα, δηλαδή τους προγραμματιστές.
- Το σχέδιο δεν πρέπει να περιέχει σφάλματα.

Τα παραπάνω, εκτός από κριτήρια, μπορούν να ακούγονται εξίσου εύκολα και σαν ευχές. Μια πρακτική σχεδίασης πρέπει να στοχεύει στην καθοδήγηση του κατασκευαστή στην παραγωγή σχεδίου λογισμικού το οποίο να πληροί στο μέγιστο δυνατό βαθμό τα παραπάνω κριτήρια. Αυτό δεν είναι πάντα εύκολο, καθότι ενδέχεται να υπάρχουν και εσωτερικές αντιφάσεις μέσα στα κριτήρια (λ.χ. αντιφάσκουσες μεταξύ τους προδιαγραφές) αλλά και άλλοι πρακτικοί λόγοι (λ.χ. απαγορευτικό από τον προϋπολογισμό κόστος πλήρους λεπτομερούς περιγραφής του σχεδίου). Η Τεχνολογία Λογισμικού παρέχει μεθοδολογίες σχεδίασης οι οποίες αποτελούν κατευθυντήριες γραμμές για τον σχεδιαστή. Συνδυάζοντας τη γνώση, την εμπειρία και τον αυτοσχεδιασμό, αλλά και με τη χρήση των κατάλληλων εργαλείων ανάπτυξης λογισμικού, ο σχεδιαστής μπορεί να αντιμετωπίσει την πρόκληση –γιατί περί πρόκλησης πρόκειται– ικανοποίησης των κριτηρίων αυτών.



## Άσκηση Ι/Κεφάλαιο 5

**Ποια είναι βασική παραδοχή που κάνει ο μηχανικός λογισμικού κατά την έναρξη της σχεδίασης λογισμικού; Ισχύει με ασφάλεια η παραδοχή αυτή;**

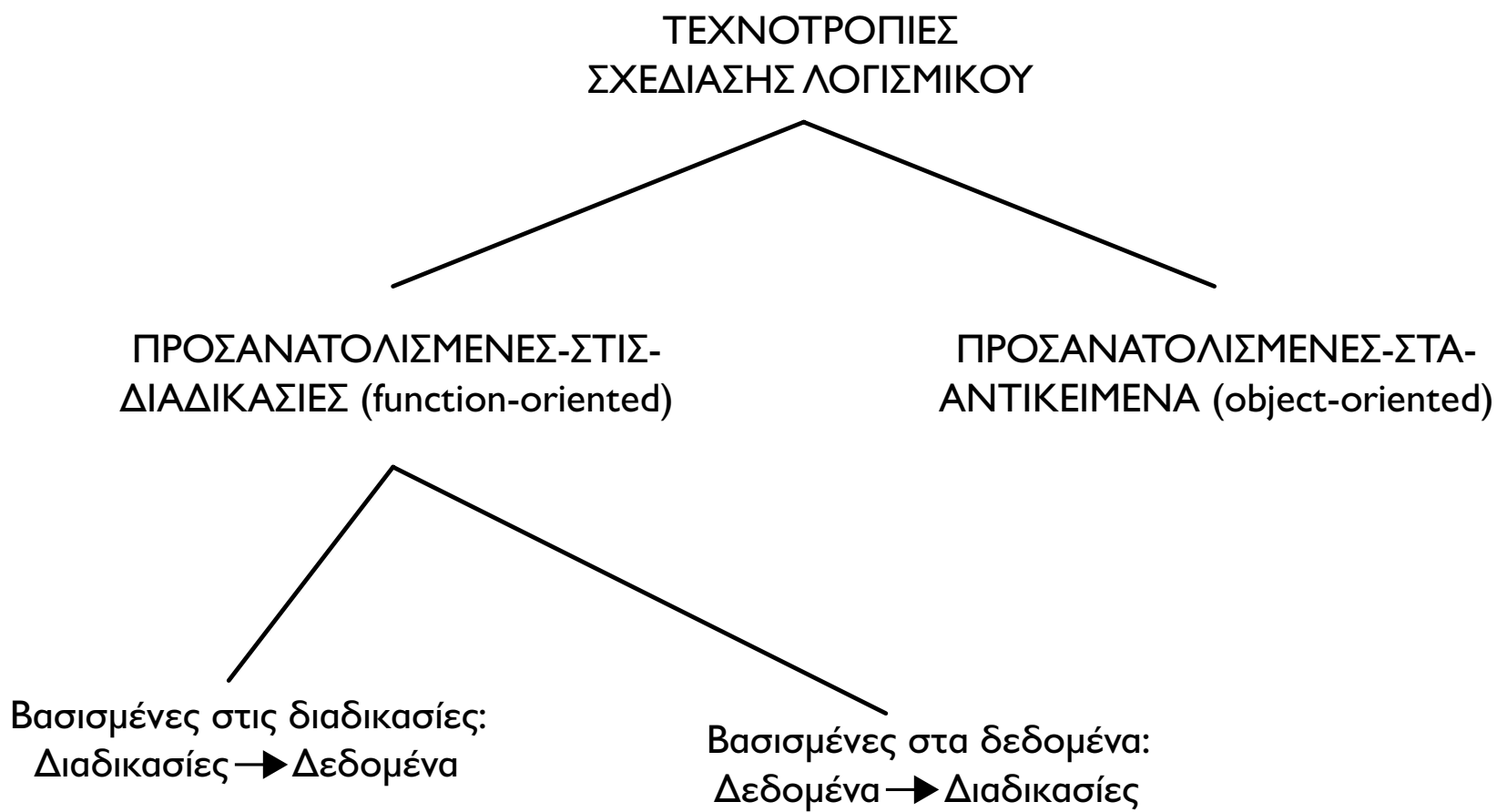
### ΕΝΟΤΗΤΑ 5.3. ΤΕΧΝΟΤΡΟΠΙΕΣ ΣΧΕΔΙΑΣΗΣ

Το πρόβλημα της σχεδίασης λογισμικού μπορεί να αντιμετωπιστεί με διάφορες στρατηγικές προσεγγίσεις. Οι διάφορες μεθοδολογίες που έχουν παρουσιαστεί μπορούν να ενταχθούν σε δύο μεγάλες κατηγορίες: τις προσανατολισμένες στις διαδικασίες (function-oriented) και τις προσανατολισμένες στα αντικείμενα (object-oriented). Μια σύντομη αναφορά στα χαρακτηριστικά των δύο κατηγοριών δίνεται στην ενότητα αυτή.

Η απάντηση στην ερώτηση «τι θα κάνει το λογισμικό;» για μεγάλο χρονικό διάστημα δινόταν με τη μορφή μιας ιεραρχίας διαδικασιών, συναρτήσεων και άλλων ενεργών μονάδων λογισμικού, η οποία επιδρούσε σε ένα σύνολο ανεξάρτητων δεδομένων. Αν και από τη δεκαετία του '60 ακόμα είχαν παρουσιαστεί και άλλες απόψεις, η προσέγγιση αυτή επικράτησε και πάνω της γεννήθηκαν διάφορες μεθοδολογίες σχεδίασης λογισμικού οι οποίες συγκρότησαν την οικογένεια μεθοδολογιών της δομημένης σχεδίασης.

Η «άλλη άποψη» προσπαθούσε να απαντήσει στο ίδιο ερώτημα χωρίς να διακρίνει τα δεδομένα από τις διαδικασίες, προτείνοντας ένα σύνολο συνεργαζόμενων οντοτήτων που περιλαμβάνουν στο ίδιο κέλυφος και δεδομένα και διαδικασίες. Οι οντότητες αυτές ονομάστηκαν «αντικείμενα» (objects). Οι μεθοδολογίες που γεννήθηκαν πάνω σε αυτή την προσέγγιση ονομάστηκαν «προσανατολισμένες στα αντικείμενα» (object-oriented). Και στις δύο περιπτώσεις ο σχεδιαστής καταπιάνεται και με τις διεργασίες και με τα δεδομένα, αποδίδοντάς τους, όμως, διαφορετική βαρύτητα σε κάθε περίπτωση. Οι δύο οικογένειες φαίνονται στο Σχήμα 5.1.

**Σχήμα 5.1** Μια ταξινόμηση των τεχνοτροπιών σχεδίασης λογισμικού.



### 5.2.1. Δομημένη σχεδίαση

Οι μεθοδολογίες που ακολουθούν αυτή την προσέγγιση προτείνουν τρόπους αποσύνθεσης του συστήματος από πάνω προς τα κάτω (top-down), σε μια ιεραρχία διαδικασιών, συναρτήσεων και άλλων ενεργών μονάδων λογισμικού. Όσο κατεβαίνει κανείς στην ιεραρχία αυτή, τόσο μεγαλύτερη λεπτομέρεια συναντά, μέχρις ότου φτάσει στις απλές δομικές μονάδες, δηλαδή τις εντολές της γλώσσας προγραμματισμού. Η προσέγγιση αυτή παρουσιάζεται στο παρόν βιβλίο.

Γνωστές μεθοδολογίες που ανήκουν στην οικογένεια αυτή έχουν προταθεί από πολλούς συγγραφείς και σχετικά μπορείτε να ανατρέξετε στη βιβλιογραφία. Οι περισσότερες των προσεγγίσεων αυτών επικεντρώνουν την προσοχή τους πρώτα στις διαδικασίες και μετά στα δεδομένα. Οι πιο σύγχρονες καθορίζουν τη δομή των διαδικασιών που επιδρούν πάνω στα δεδομένα με βάση τη δομή των δεδομένων αυτών και για τον λόγο αυτό χαρακτηρίζονται ως «βασισμένες στα δεδομένα» και συγγενεύουν εν μέρει με τις προσανατολισμένες στα αντικείμενα τεχνοτροπίες.

### 5.2.2. Αντικειμενοστρεφής σχεδίαση

Η αντικειμενοστρεφής (object-oriented) προσέγγιση ακολουθεί ένα διαφορετικό δρόμο: αντί το σύστημα να θεωρείται ως μια ιεραρχία διαδικασιών ανεξάρτητων από τα δεδομένα, θεωρείται ως μια συλλογή οντοτήτων, καθεμία εκ των οποίων περικλείει και διαδικασίες και δεδομένα. Η προσέγγιση βασίζεται στην ιδέα ότι στον πραγματικό κόσμο δεδομένα και διαδικασίες μπορούν να ιδωθούν ενιαία με βάση το πεδίο ευθύνης κάποιων οντοτήτων που ονομάζονται «αντικείμενα». Κάθε αντικείμενο παρέχει στο περιβάλλον του ένα σύνολο υπηρεσιών της ευθύνης του. Η συνεργασία του συνόλου των αντικειμένων του πεδίου μιας εφαρμογής λογισμικού παράγει το επιθυμητό αποτέλεσμα.

Μερικές από τις πιο γνωστές προσεγγίσεις που ανήκουν στην κατηγορία αυτή προτάθηκαν από τους Meyer (1988), Rumbaugh (1992), Jacobson (1993) και Booch (1994). Για μεγάλο χρονικό διάστημα επικρατούσε μια σύγχυση σε επίπεδο ορολογίας και συμβολισμών στην οικογένεια της

αντικειμενοστρεφούς ανάλυσης και σχεδίασης. Τα τελευταία χρόνια η σύγχυση αυτή φαίνεται να περιορίζεται με την εμφάνιση «συγκωνευμένων» μεθοδολογιών και ενοποιημένων συμβολισμών.

### Δραστηριότητα I/Κεφάλαιο 5

Περιγράψτε σε μια παράγραφο που δεν ξεπερνά τις δέκα γραμμές ποια είναι η βασική διαφορά των δύο προσεγγίσεων σχεδίασης που αναφέρθηκαν.

## ΕΝΟΤΗΤΑ 5.2. ΑΝΤΙΚΕΙΜΕΝΟ ΚΑΙ ΑΠΟΤΕΛΕΣΜΑΤΑ ΤΗΣ ΣΧΕΔΙΑΣΗΣ

Στην ενότητα αυτή θα καθοριστούν τα αντικείμενα εργασίας κατά τη σχεδίαση λογισμικού, δηλαδή το ποια σχέδια πρέπει να κατασκευαστούν και το τι περιγράφει καθένα από αυτά. Επίσης, θα παρουσιαστεί ένας τρόπος παράστασης των αποτελεσμάτων της σχεδίασης με δομημένο κείμενο, ανάλογος με αυτόν του εγγράφου προδιαγραφών των απαιτήσεων από το λογισμικό. Η αναφορά θα ακολουθήσει την προσέγγιση της δομημένης σχεδίασης.

### 5.3.1. Εισαγωγή

Στον ορισμό που δόθηκε στην προηγούμενη ενότητα, το σχέδιο λογισμικού χαρακτηρίζεται ως «η περιγραφή των μονάδων που αποτελούν το λογισμικό, των συσχετίσεων μεταξύ τους, της διάταξής τους, καθώς και της εσωτερικής τους λεπτομέρειας». Προκειμένου να γίνει δυνατή η περιγραφή αυτή, πρέπει να αντιμετωπίσουμε το πρόβλημα σε τέσσερα επίπεδα ως ακολούθως:

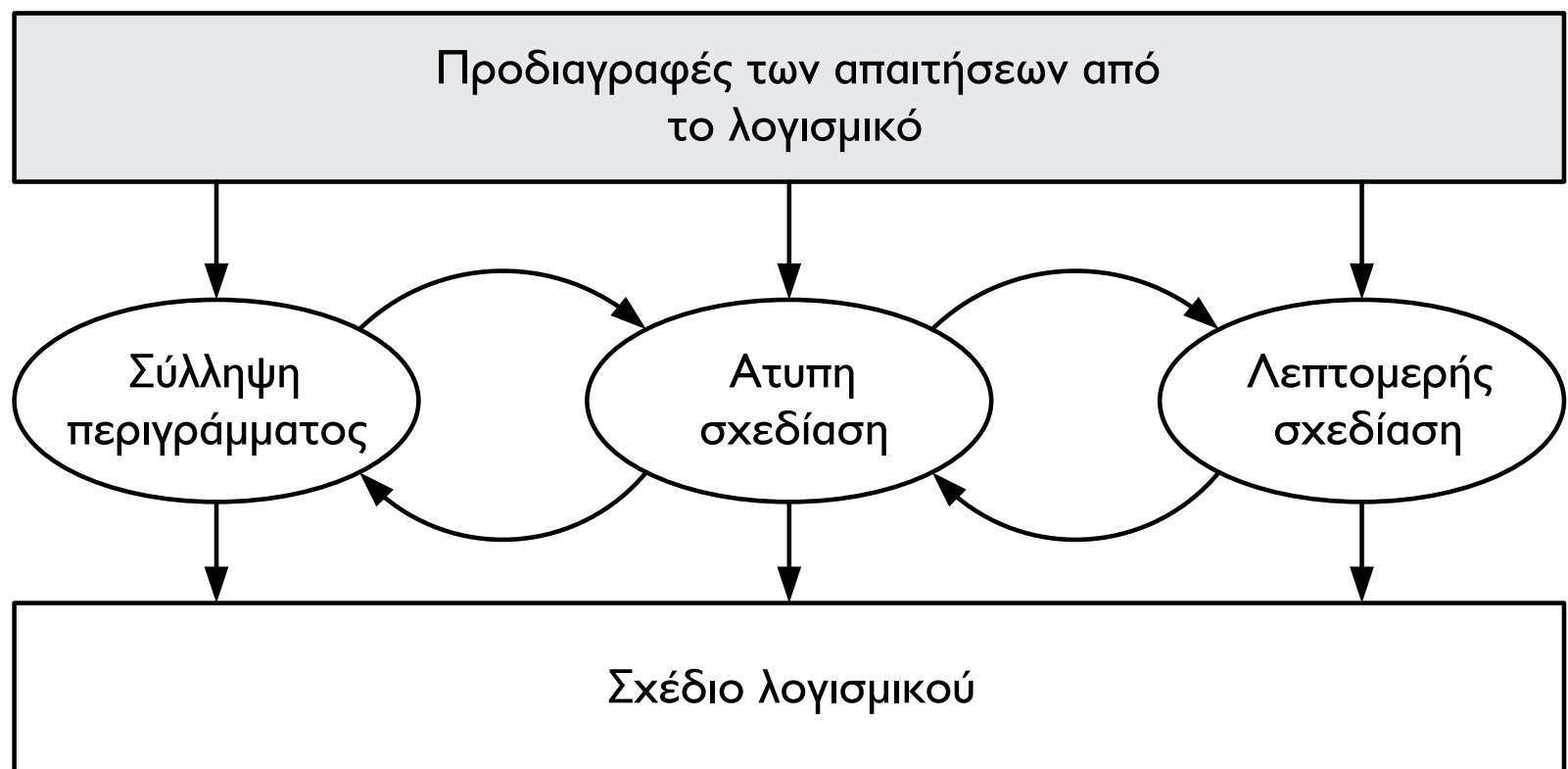
- **Αρχιτεκτονική σχεδίαση.** Αφορά τον καθορισμό του ποιες είναι οι μονάδες που συγκροτούν το σύστημα λογισμικού και πώς αυτές

ανατίθενται για εκτέλεση (διατάσσονται) στις υπολογιστικές μονάδες που είναι διαθέσιμες.

- **Σχεδίαση διεπαφών** (interfaces). Αφορά τον καθορισμό της επικοινωνίας των μονάδων μεταξύ τους, με άλλα συστήματα λογισμικού, με άλλες συσκευές, καθώς και με τον άνθρωπο. Όταν λέμε «καθορισμός επικοινωνίας», εννοούμε την περιγραφή του ποια μονάδα επικοινωνεί με ποια, με ποιες παραμέτρους, καθώς και με όποιο άλλο στοιχείο απαιτείται για να περιγραφεί επαρκώς κατά περίπτωση η επικοινωνία.
- **Λεπτομερής σχεδίαση μονάδων**. Αφορά τον καθορισμό της εσωτερικής δομής κάθε μονάδας λογισμικού, προκειμένου αυτή να ικανοποιεί, αφενός, τις λειτουργικές απαιτήσεις και, αφετέρου, να συνεργάζεται με τις άλλες μονάδες, όπως έχει καθοριστεί κατά την αρχιτεκτονική και τη σχεδίαση δεδομένων.
- **Σχεδίαση δεδομένων**. Πρόκειται για τη λεπτομερή σχεδίαση των δεδομένων των οποίων η τήρηση αποτελεί απαίτηση από το λογισμικό και η οποία έχει τεθεί στη φάση προδιαγραφής των απαιτήσεων.

Τα προϊόντα που έχουν παραχθεί κατά τη φάση της προδιαγραφής των απαιτήσεων από το λογισμικό αποτελούν την είσοδο στην εργασία της σχεδίασης. Σε όλα της τα επίπεδα, η σχεδίαση λογισμικού είναι μια εργασία που προχωρά από τη μικρότερη στη μεγαλύτερη λεπτομέρεια, όπως φαίνεται στο Σχήμα 5.2.

**Σχήμα 5.2** Εξέλιξη της σχεδίασης λογισμικού.



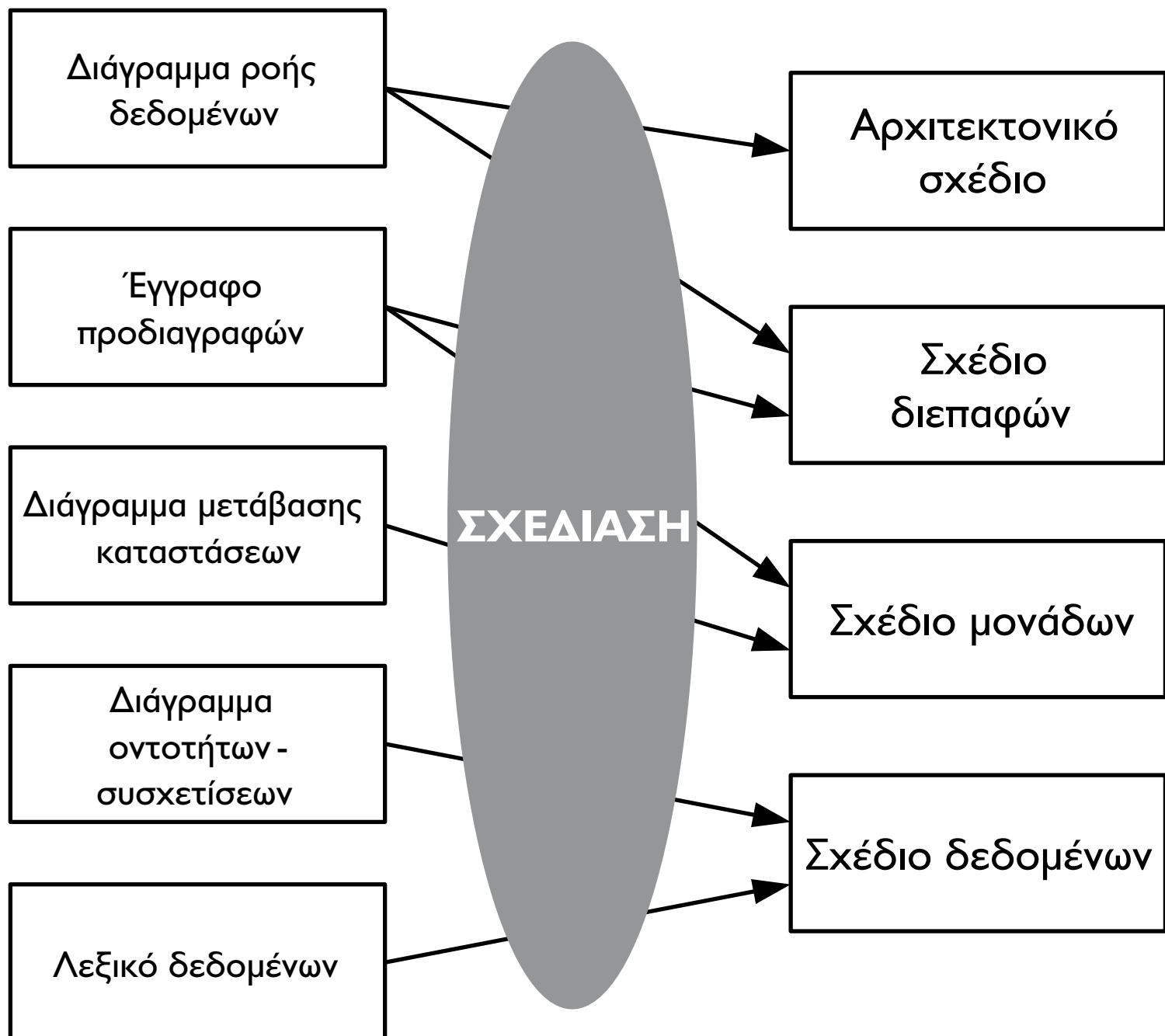
Συμβολισμοί :

Προϊόν

Εργασία

Στο Σχήμα 5.3 φαίνεται μια συσχέτιση μεταξύ των προϊόντων σχεδίασης που προκύπτουν από την εκτέλεση των εργασιών στα τέσσερα επίπεδα που αναφέρθηκαν με τα προϊόντα της φάσης προσδιορισμού των προδιαγραφών.

**Σχήμα 5.3** Από τα προϊόντα προδιαγραφών των απαιτήσεων στα προϊόντα σχεδίασης λογισμικού.

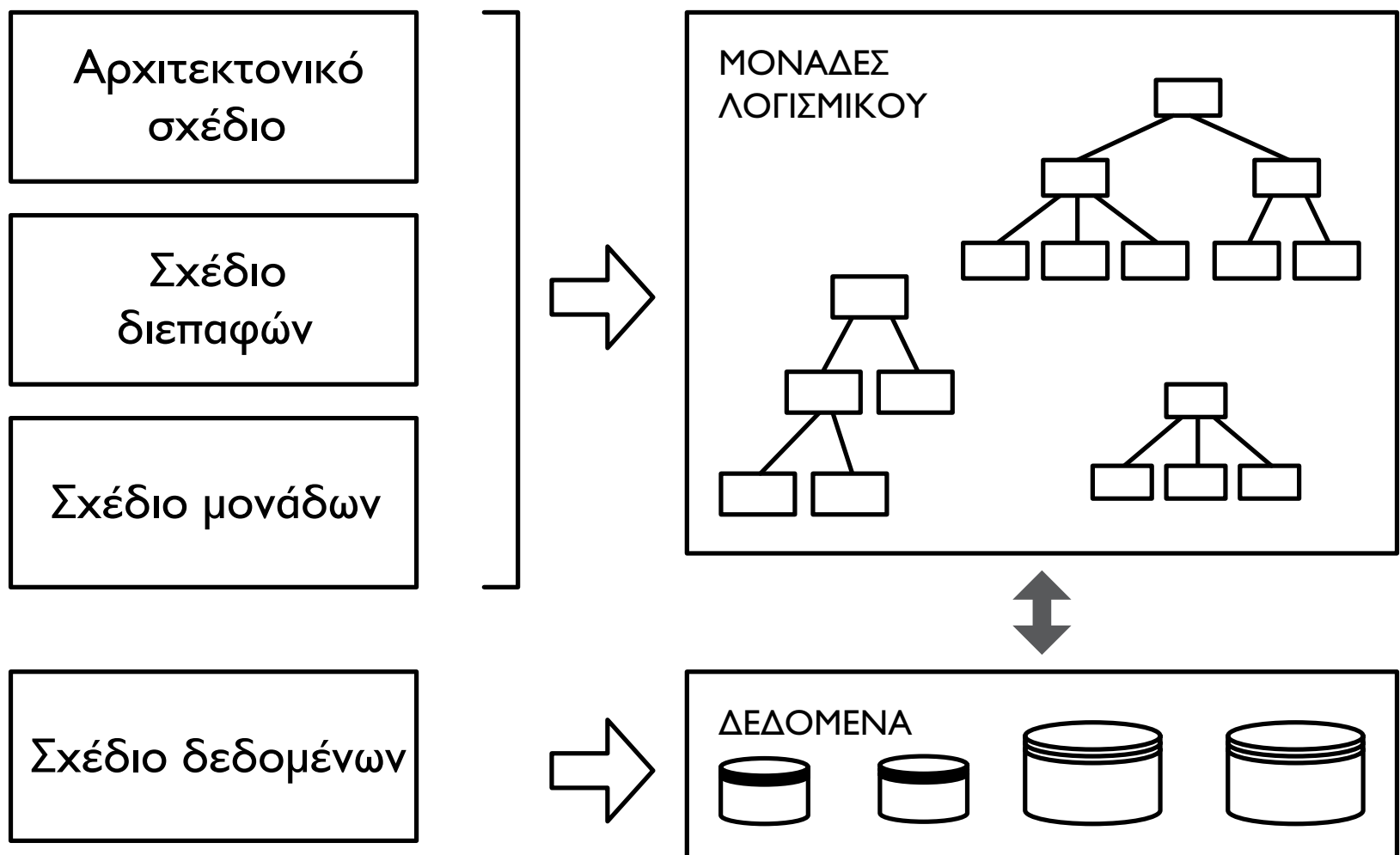




Όπως φαίνεται και στο Σχήμα 5.3, στη δομημένη ανάλυση και σχεδίαση μπορούμε να διακρίνουμε τη σχεδίαση των δεδομένων από τη σχεδίαση που αφορά τις μονάδες προγράμματος. Η σχεδίαση των δεδομένων φαίνεται να σχετίζεται σχεδόν αποκλειστικά με το διάγραμμα οντοτήτων – συσχετίσεων και με το λεξικό δεδομένων, ενώ η σχεδίαση μονάδων (αρχιτεκτονική, επικοινωνία, δομή) σχετίζεται με το σύνολο των υπόλοιπων αποτελεσμάτων των προδιαγραφών. Αποφεύγουμε τον ορισμό της έννοιας «μονάδα λογισμικού», διότι αυτός εξαρτάται από τη γλώσσα προγραμματισμού που χρησιμοποιείται. Οι συναρτήσεις (functions), οι υπορουτίνες (subroutines), οι μονάδες (units) και οι διαδικασίες (procedures) είναι οι συνηθέστεροι όροι που χρησιμοποιούνται από τις γλώσσες προγραμματισμού για να περιγράψουν μονάδες λογισμικού.

Στο Σχήμα 5.4 φαίνεται η ανεξαρτησία των δεδομένων από τις μονάδες λογισμικού η οποία, όπως τονίστηκε, είναι ουσιώδες χαρακτηριστικό της δομημένης ανάλυσης και σχεδίασης. Από το αρχιτεκτονικό σχέδιο, το σχέδιο διεπαφών και το λεπτομερές σχέδιο μονάδων τελικά θα κατασκευαστεί μια δομή ενεργών συστατικών (μονάδων) λογισμικού τα οποία θα επιδρούν πάνω σε κάποια ανεξάρτητα δεδομένα.

**Σχήμα 5.4** Δεδομένα και μονάδες λογισμικού στη δομημένη ανάλυση και σχεδίαση.



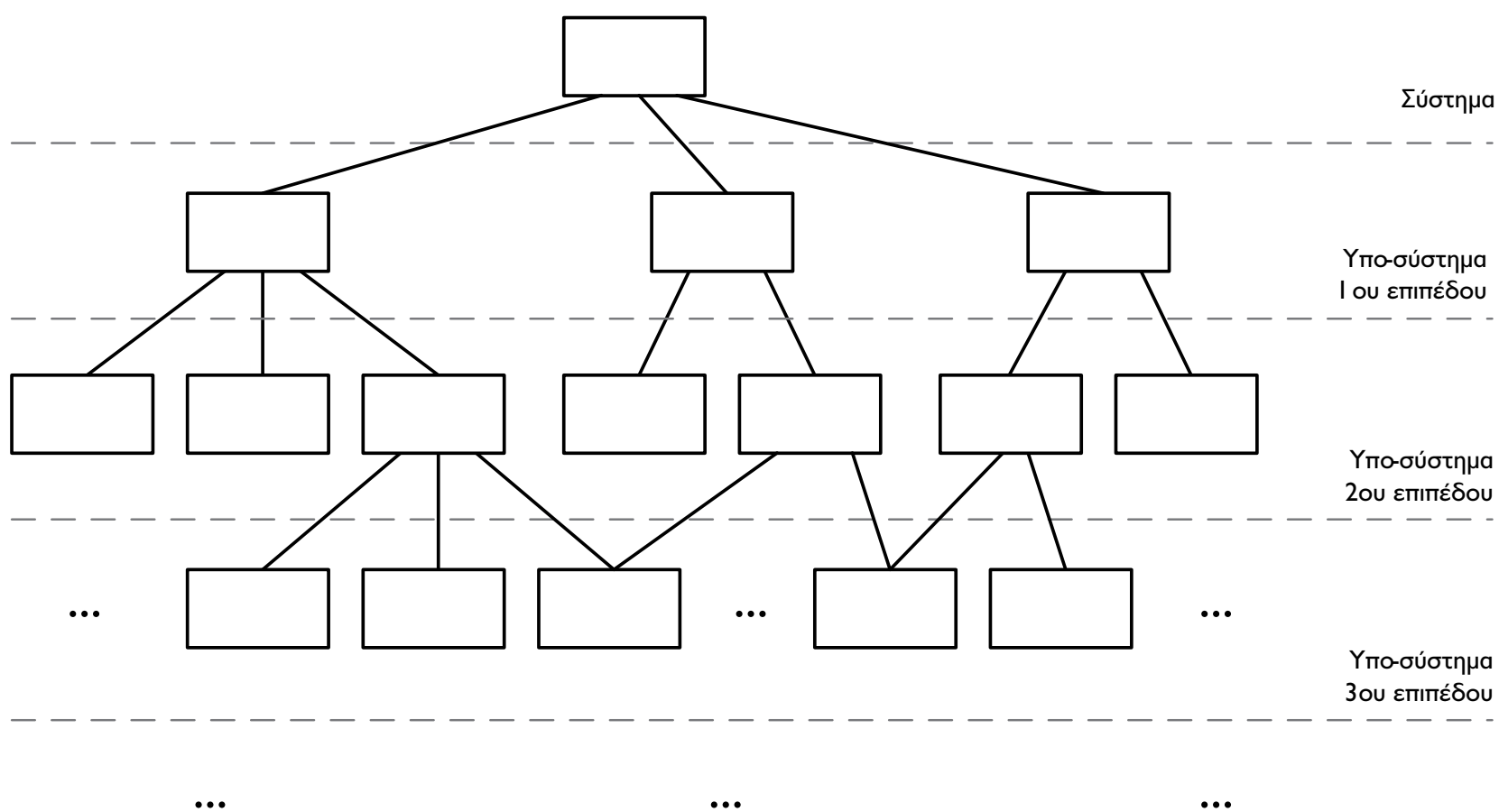
### 5.3.2. Αρχιτεκτονική σχεδίαση.

Όπως αναφέρθηκε, κατά την αρχιτεκτονική σχεδίαση προσδιορίζεται το ποιες είναι οι μονάδες που αποτελούν το λογισμικό και πώς σχετίζονται μεταξύ τους. Ο προσδιορισμός αυτός αναφέρεται και ως σχεδίαση συστήματος. Στη δομημένη σχεδίαση η εργασία αυτή γίνεται λαμβάνοντας ως είσοδο τα διαγράμματα ροής δεδομένων. Σε αυτά εφαρμόζεται ένα σύνολο κανόνων και καθορίζονται οι μονάδες λογισμικού που είναι αναγκαίες για την υλοποίηση των μετασχηματισμών και των ροών δεδομένων. Αναλυτική περιγραφή της διαδικασίας αυτής θα δοθεί στην υποενότητα 5.4.2.

Η αρχιτεκτονική δομή του λογισμικού μπορεί να ιδωθεί σε πολλά επίπεδα λεπτομέρειας. Στο Σχήμα 5.5 φαίνεται μία άποψη, όπου το λογισμικό θεωρείται στο πρώτο επίπεδο ως ένα σύνθετο σύστημα το οποίο αποτελείται από υποσυστήματα (δεύτερο επίπεδο), καθένα εκ των οποίων αποτελείται από υποσυστήματα δεύτερης τάξης (τρίτο επίπεδο) κ.ο.κ., μέχρις ότου φτάσουμε στις ατομικές μονάδες λογισμικού (συναρτήσεις, υπορουτίνες κ.λπ.).

Η αρχιτεκτονική λογισμικού καθορίζει ποιες θα είναι αυτές, και στη συνέχεια το λεπτομερές σχέδιο μονάδων θα καθορίσει πώς ακριβώς θα υλοποιηθούν. Σε όσο χαμηλότερο επίπεδο βρισκόμαστε, τόσο πιθανότερο είναι να αναγνωρίζονται κοινοί παράγοντες, δηλαδή μονάδες λογισμικού που ανήκουν σε περισσότερα του ενός υποσυστήματα του αμέσως παραπάνω επιπέδου. Το φαινόμενο εμφανίζεται κυρίως μετά από την εκτέλεση της συναρτησιακής αποσύνθεσης (functional decomposition) στην οποία θα αναφερθούμε κατά την περιγραφή της λεπτομερούς σχεδίασης μονάδων. Στο Σχήμα 5.5 τέτοιες μονάδες σημειώνονται με γκρι χρώμα.

**Σχήμα 5.5** Επίπεδα αρχιτεκτονικού σχεδίου λογισμικού.



Οι μονάδες που αποτελούν το λογισμικό μπορεί να ανατίθενται προς εκτέλεση σε πολλούς διατιθέμενους υπολογιστικούς πόρους (υπολογιστικά συστήματα, μονάδες επεξεργασίας). Στην απλούστερη περίπτωση, το λογισμικό είναι κατασκευασμένο ώστε να τρέχει εξ ολοκλήρου σε ένα μόνο υπολογιστικό σύστημα. Στη γενικότερη περίπτωση, τα υποσυστήματα του λογισμικού μπορούν να ανατίθενται σε διαφορετικούς υπολογιστικούς πόρους.

### 5.3.3. Σχεδίαση διεπαφών

Καμία μονάδα λογισμικού δεν είναι ανεξάρτητη. Είτε χρησιμοποιεί άλλες μονάδες προκειμένου να επιτελέσει τη λειτουργία για την οποία προορίζεται είτε χρησιμοποιείται από άλλες μονάδες για τον ίδιο σκοπό. Η διατύπωση ότι μία μονάδα λογισμικού χρησιμοποιεί άλλες μονάδες δηλώνει την κλήση προγραμμάτων, υπορουτινών, συναρτήσεων και άλλων δομικών στοιχείων λογισμικού. Η κλήση μιας μονάδας B από μια μονάδα A σημαίνει:

- Ενεργοποίηση της μονάδας B, δηλαδή η ροή του προγράμματος μεταφέρεται στη μονάδα B και εκτελούνται οι εντολές που περιέχονται σε αυτή.
- Επικοινωνία της μονάδας A με τη B μέσω παραμέτρων που περνά η A στη B και ενδεχομένως και αντίστροφα.

Κατά τη σχεδίαση διεπαφών (interface design) καθορίζονται οι λεπτομέρειες του περάσματος των παραμέτρων αυτών σε όλα τα επίπεδα επικοινωνίας. Οι μονάδες λογισμικού δεν επικοινωνούν μόνο μεταξύ τους αλλά και με εξωτερικά συστήματα ή συσκευές, καθώς και με τον άνθρωπο. Συγκεντρώνοντας το σύνολο των απαιτήσεων από τη σχεδίαση διεπαφών, καταλήγουμε ότι κατά την εργασία αυτή θα πρέπει να καθοριστούν τα ακόλουθα:

- Ο αριθμός και ο τύπος των παραμέτρων κατά την κλήση μονάδων λογισμικού.
- Το είδος και οι λεπτομέρειες της επικοινωνίας με άλλα συστήματα λογισμικού.
- Το είδος και οι λεπτομέρειες της επικοινωνίας με εξωτερικές συσκευές.
- Η επικοινωνία του λογισμικού με τον άνθρωπο.

Οι παράμετροι κατά την επικοινωνία με άλλες μονάδες λογισμικού (στην ίδια εφαρμογή) καθορίζονται κατά την αρχιτεκτονική και τη λεπτομερή σχεδίαση και περιγράφονται στα διαγράμματα δομής προγράμματος και το λεπτομερές σχέδιο μονάδων, στα οποία θα αναφερθούμε σε επόμενες παραγράφους. Οι εξωτερικές διεπαφές περιγράφονται αυτοτελώς με τον προσφορότερο κατά περίπτωση τρόπο, ανάλογα με τα εξωτερικά συστήματα λογισμικού ή τις εξωτερικές συσκευές. Η σχεδίαση της διεπαφής με τον άνθρωπο είναι ένα ιδιαίτερα πολύπλοκο πρόβλημα στο οποίο, εκτός από μηχανικούς λογισμικού, μπορεί να έχουν λόγο και άλλες ειδικότητες, όπως κοινωνιολόγοι και ψυχολόγοι.

#### **5.3.4. Λεπτομερής σχεδίαση μονάδων**

Το πιο λεπτομερές επίπεδο της σχεδίασης είναι η σχεδίαση μονάδων. Πρόκειται για το σημείο όπου πρέπει να καθοριστούν όλες οι κατασκευαστικές λεπτομέρειες που απαιτούνται για τη δημιουργία του πηγαίου κώδικα για όλες τις μονάδες λογισμικού. Η γνώση της ύπαρξης μίας μονάδας και των χαρακτηριστικών επικοινωνίας αυτής με άλλες μονάδες δεν επαρκεί, στη γενική περίπτωση, για την κατασκευή του πηγαίου κώδικα της ίδιας της μονάδας. Σε τετριμμένες περιπτώσεις η λεπτομερής περιγραφή της εσωτερικής δομής δεν είναι αναγκαία για την κατασκευή της μονάδας. Στις περισσότερες όμως των περιπτώσεων αυτό δεν ισχύει και η λεπτομερής σχεδίαση μονάδων είναι αναγκαία.

Το υλικό που χρησιμοποιείται ως είσοδος στη φάση της λεπτομερούς σχεδίασης μονάδων είναι το αρχιτεκτονικό σχέδιο, το σχέδιο διεπαφών, το έγγραφο προδιαγραφών των απαιτήσεων από το λογισμικό, καθώς και το διάγραμμα μετάβασης καταστάσεων. Τα δύο πρώτα έχουν κατασκευαστεί κατά τη σχεδίαση, ενώ τα δύο τελευταία κατασκευάστηκαν κατά την προδιαγραφή των απαιτήσεων και περιέχουν περιγραφή της λειτουργικής συμπεριφοράς του λογισμικού με τη μορφή απαιτήσεων.

Κατά τη λεπτομερή σχεδίαση, για κάθε μονάδα θα δοθεί ένα περίγραμμα – ομοίωμα της δομής της, χρήσιμο για την κατασκευή της. Αποτέλεσμα της λεπτομερούς σχεδίασης είναι το λεπτομερές σχέδιο μονάδων, το οποίο

στη συνέχεια θα δοθεί στους προγραμματιστές που θα συγγράψουν τον πηγαίο κώδικα.

### 5.3.5. Σχεδίαση δεδομένων

Η σχεδίαση δεδομένων θεωρείται ως μία από τις σημαντικότερες διαδικασίες στην ανάπτυξη λογισμικού. Η σύλληψη της δομής και των συσχετίσεων των δεδομένων μπορεί να καθορίζει πολλά από τα χαρακτηριστικά των λειτουργικών μονάδων λογισμικού. Στη δομημένη ανάλυση και σχεδίαση ο προσδιορισμός των απαιτήσεων και η λεπτομερής σχεδίαση των δεδομένων, από τη μία, και η σχεδίαση των μονάδων λογισμικού, από την άλλη, είναι δύο διαφορετικές διακριτές εργασίες.

Προκειμένου να μπορεί να κατασκευαστεί με πληρότητα το λεπτομερές σχέδιο μονάδων λογισμικού, είναι απαραίτητο να διατίθεται το λεπτομερές σχέδιο δεδομένων. Αυτό ισχύει για τις περιπτώσεις λειτουργικών μονάδων λογισμικού που επιδρούν με οποιοδήποτε τρόπο πάνω σε δεδομένα. Έχοντας, λοιπόν, υπόψη το λεξικό δεδομένων και το διάγραμμα οντοτήτων – συσχετίσεων, ο σχεδιαστής λογισμικού:

- πραγματοποιεί βελτιστοποιήσεις στο μοντέλο οντοτήτων – συσχετίσεων με σκοπό τη βελτιστοποίηση των επιδόσεων, την πληροφοριακή πληρότητα και την εξάλειψη πλεονάζουσας (redundant) πληροφορίας,
- καθορίζει τη δομή κάθε πίνακα (πεδία και τύποι αυτών) στο φυσικό επίπεδο,
- καθορίζει τις δυνατές απόψεις των δεδομένων στο λογικό επίπεδο (views).

Με το αντικείμενο της λεπτομερούς σχεδίασης δεδομένων ασχολείται η γνωστική περιοχή των βάσεων δεδομένων. Το αποτέλεσμα της σχεδίασης δεδομένων είναι η λεπτομερής περιγραφή των οντοτήτων και των συσχετίσεων αυτών σε επίπεδο που να είναι εφικτή η υλοποίηση της βάσης δεδομένων.

### 5.3.6. Το έγγραφο περιγραφής του σχεδίου του λογισμικού

Σε αναλογία με το έγγραφο προδιαγραφών των απαιτήσεων από το λογισμικό, είναι χρήσιμο το σχέδιο του λογισμικού να αποτυπώνεται με χρήση ενός δομημένου εγγράφου. Το έγγραφο αυτό αποτελεί ένα ενιαίο δομικό κέλυφος στο οποίο ενσωματώνονται όλα τα επιμέρους προϊόντα της σχεδίασης με τη μορφή διαγραμμάτων αλλά και με τη μορφή κειμένων ή άλλων τρόπων περιγραφής σχεδίου τους οποίους θα αναφέρουμε στη συνέχεια. Μια γενική δομή του εγγράφου περιγραφής του σχεδίου του λογισμικού προτείνεται από το IEEE και παρατίθεται στο Σχήμα 5.6.



## **I. Εισαγωγή**

- I.1 Γενική περιγραφή
- I.2 Εμβέλεια
- I.3 Ορισμοί και ακρωνύμια
- I.4 Αναφορές

## **2. Περιγραφή αποσύνθεσης του λογισμικού**

- 2.1 Αποσύνθεση σε μονάδες
  - 2.1.1 Περιγραφή μονάδας 1
  - 2.1.2 Περιγραφή μονάδας 2
  - ...
- 2.2 Αποσύνθεση σε ταυτόχρονες διεργασίες
  - 2.2.1 Περιγραφή διεργασίας 1
  - 2.2.2 Περιγραφή διεργασίας 2
  - ...
- 2.3 Αποσύνθεση δεδομένων
  - 2.3.1 Περιγραφή οντότητας δεδομένων 1
  - 2.3.2 Περιγραφή οντότητας δεδομένων 2
  - ...

## **3. Περιγραφή εξαρτήσεων**

- 3.1 Εξαρτήσεις μεταξύ μονάδων
- 3.2 Εξαρτήσεις μεταξύ διεργασιών
- 3.3 Εξαρτήσεις μεταξύ δεδομένων

## **4. Περιγραφή διεπαφών**

- 4.1 Διεπαφές μονάδων
- 4.2 Διεπαφές διεργασιών
- 4.3 Διεπαφές χρήστη
- 4.4 Εξωτερικές διεπαφές
  - 4.4.1 Συστήματα λογισμικού
  - 4.4.2 Συσκευές

## **5. Λεπτομερές σχέδιο μονάδων**

- 5.1 Λεπτομερές σχέδιο μονάδας 1
- 5.2 Λεπτομερές σχέδιο μονάδας 2
- ...

## **6. Λεπτομερές σχέδιο δεδομένων**

- 6.1 Οντότητα δεδομένων 1
- 6.2 Οντότητα δεδομένων 2
- ...

## **Παραρτήματα**

**Σχήμα 5.6** Το έγγραφο περιγραφής του σχεδίου του λογισμικού (προτεινόμενη δομή σύμφωνα με το IEEE).

## Άσκηση 2/Κεφάλαιο 5

Εντοπίστε τις παραγράφους του εγγράφου περιγραφής του σχεδίου του λογισμικού οι οποίες συμπληρώνονται σε καθένα από τα τέσσερα επίπεδα της σχεδίασης που αναφέρθηκαν, δηλαδή στην αρχιτεκτονική σχεδίαση, στη σχεδίαση διεπαφών, στη λεπτομερή σχεδίαση μονάδων και στη σχεδίαση δεδομένων.

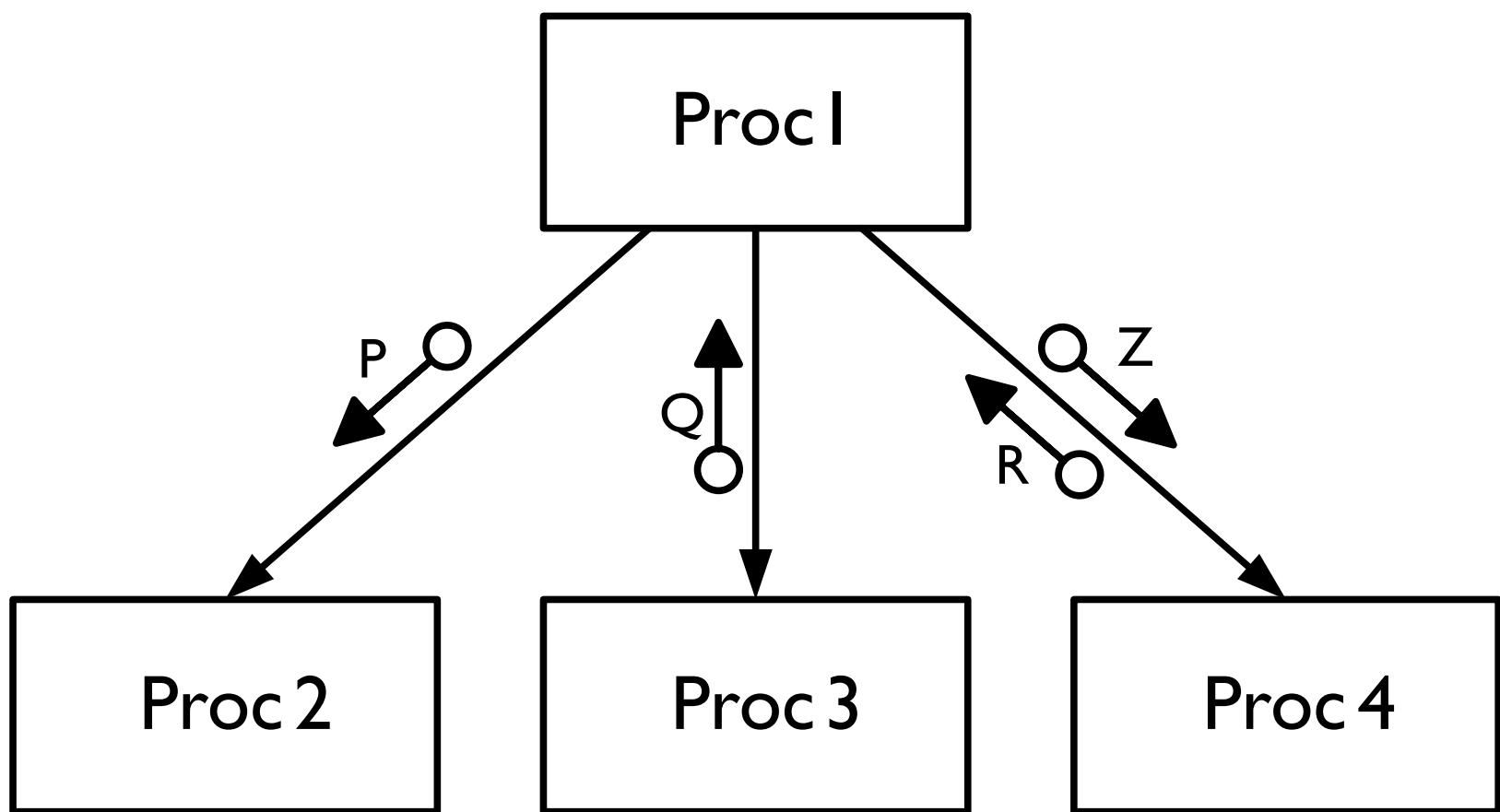
### ΕΝΟΤΗΤΑ 5.4. ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΣΧΕΔΙΑΣΗ

Στην ενότητα αυτή θα περιγραφεί η διαδικασία κατασκευής του αρχιτεκτονικού σχεδίου συστήματος με βάση το διάγραμμα ροής δεδομένων, η οποία ακολουθείται στη δομημένη σχεδίαση.

#### 5.4.1. Ορισμοί

Η μέθοδος της δομημένης σχεδίασης βασίζεται στα διαγράμματα ροής δεδομένων από τα οποία, με κάποιο συστηματικό αλλά όχι αυτόματο τρόπο, παράγεται το αρχιτεκτονικό σχέδιο του λογισμικού. Το σχέδιο αυτό αποτυπώνεται με τη βοήθεια ενός διαγράμματος που ονομάζεται «διάγραμμα δομής προγράμματος», όπως αυτό που φαίνεται στο Σχήμα 5.7. Στο διάγραμμα δομής προγράμματος οι μονάδες λογισμικού παριστάνονται με ένα παραλληλόγραμμο. Η κλήση της μονάδας B από τη μονάδα A υποδηλώνεται με ένα βέλος που συνδέει την A με τη B. Το πέρασμα παραμέτρων προς αμφότερες υποδηλώνεται με ένα βέλος και έναν κύκλο στο άκρο της αρχής του, σημειωμένο σε διεύθυνση παράλληλη με τη διεύθυνση του βέλους που περιγράφει την κλήση, το οποίο φέρει το όνομα της παραμέτρου.

**Σχήμα 5.7** Συμβολισμοί διαγραμμάτων δομής προγράμματος.



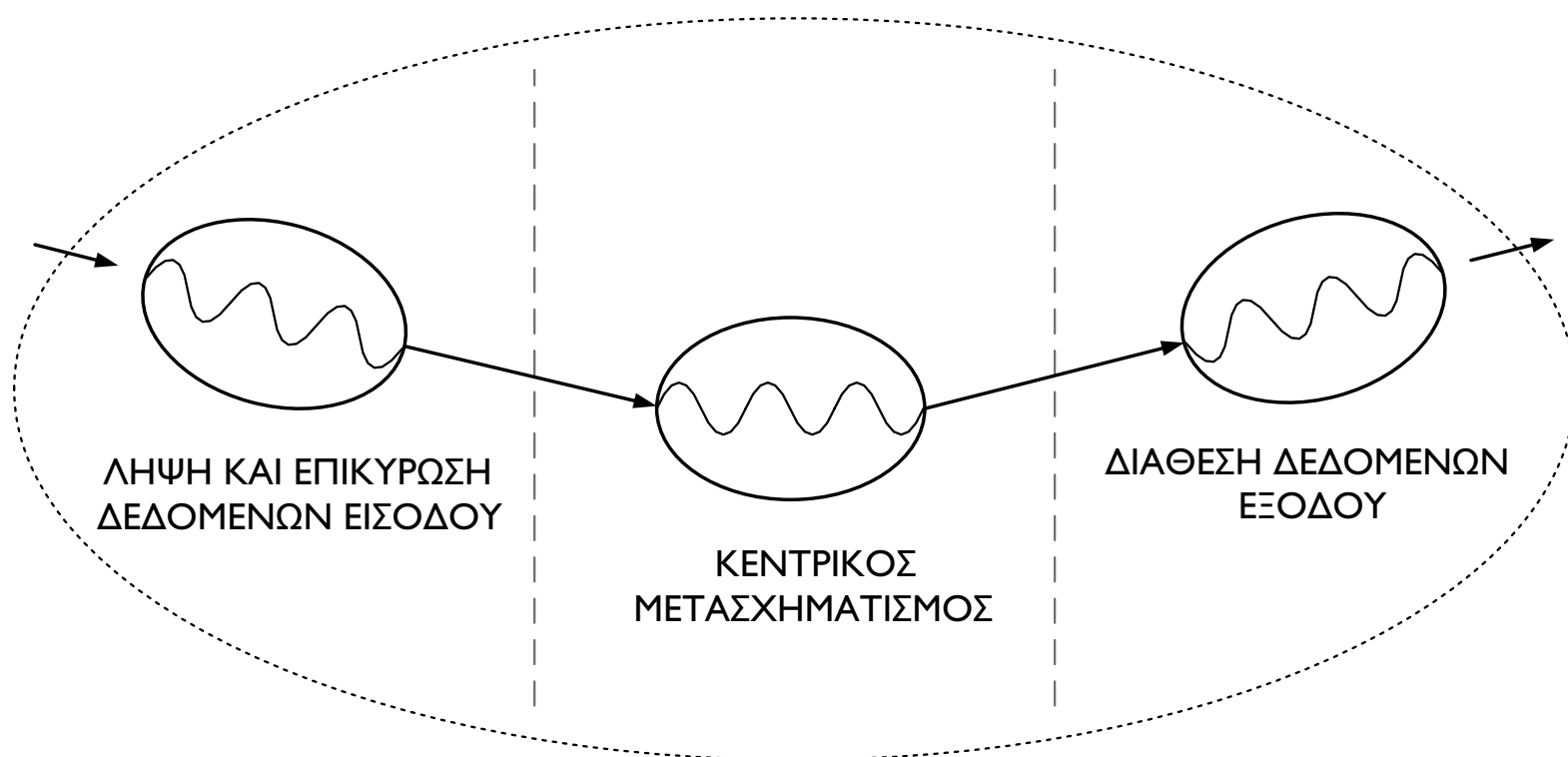
Η διαδικασία εκτελείται για καθένα από τα διαγράμματα ροής δεδομένων του συστήματος. Στα διαγράμματα αυτά εντοπίζονται δύο τύποι χαρακτηριστικών περιοχών, οι *κεντρικοί μετασχηματισμοί* και τα *κέντρα δοσοληψιών*. Όταν εντοπιστεί μια τέτοια περιοχή, δημιουργείται ή συμπληρώνεται ένα επίπεδο του διαγράμματος δομής προγράμματος και η διαδικασία επαναλαμβάνεται μέχρις ότου να εξαντληθεί το διάγραμμα ροής δεδομένων.

### **Κεντρικός μετασχηματισμός:**

Ως κεντρικός μετασχηματισμός ορίζεται μια περιοχή ενός διαγράμματος ροής δεδομένων, όπου οι μετασχηματισμοί που ανήκουν σε αυτή μπορεί να θεωρηθούν ότι μετατρέπουν δεδομένα εισόδου, προκειμένου να δημιουργήσουν νέα δεδομένα εξόδου. Οι εργασίες που προηγούνται του κεντρικού μετασχηματισμού προσάγουν τα δεδομένα εισόδου σε αυτόν, ενώ οι εργασίες που έπονται αυτού καταναλώνουν τα δεδομένα εξόδου.

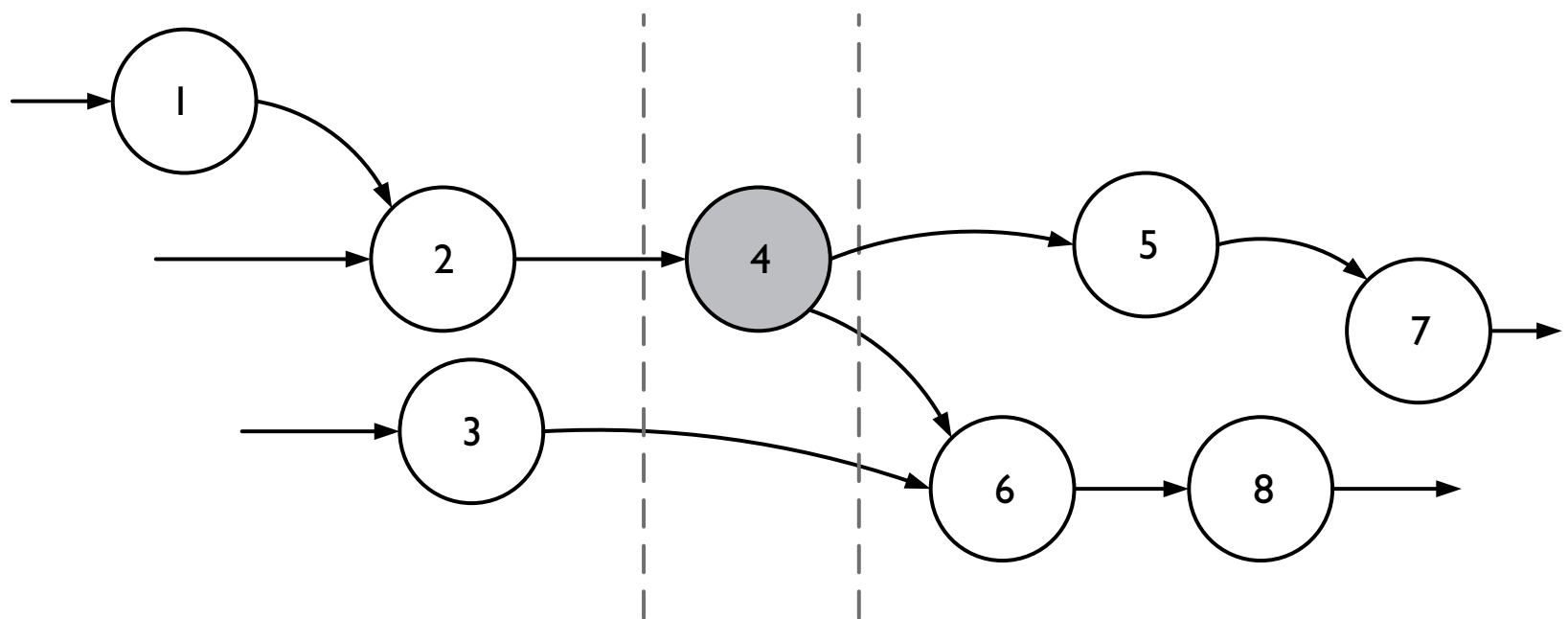
Στο Σχήμα 5.8 φαίνεται ένας κεντρικός μετασχηματισμός ως εντοπισμός χαρακτηριστικών περιοχών σε ένα υποσύνολο ενός διαγράμματος ροής δεδομένων. Στον κεντρικό μετασχηματισμό μπορεί να ανήκουν περισσότεροι του ενός μετασχηματισμοί και το ίδιο ισχύει για τα αριστερά και δεξιά τμήματα του διαγράμματος. Αυτό σημαίνει ότι ο κεντρικός μετασχηματισμός, η προετοιμασία δεδομένων εισόδου και η προετοιμασία δεδομένων εξόδου μπορούν να αντιστοιχούν σε σύνθετα τμήματα του διαγράμματος ροής δεδομένων και όχι σε απλούς μετασχηματισμούς.

**Σχήμα 5.8** Η έννοια του κεντρικού μετασχηματισμού.



Δεν υπάρχει αυτόματος τρόπος για τον εντοπισμό των κεντρικών μετασχηματισμών. Η εμπειρία, ο αυτοσχεδιασμός και η διαίσθηση του μηχανικού λογισμικού έχουν και εδώ τον πρώτο λόγο. Επίσης, η επιλογή ενός κεντρικού μετασχηματισμού δεν είναι μοναδική. Αυτό σημαίνει ότι στο ίδιο διάγραμμα ροής δεδομένων δύο ή περισσότερες περιοχές μπορούν να χαρακτηριστούν ως κεντρικός μετασχηματισμός χωρίς να είναι απαραίτητα λάθος ο ένας από τους δύο χαρακτηρισμούς. Η σύλληψη και η λεπτομέρεια του διαγράμματος ροής δεδομένων παίζουν, όπως είναι φανερό, καθοριστικό ρόλο στο σημείο αυτό. Στο Σχήμα 5.9 φαίνεται ένα διάγραμμα ροής δεδομένων στο οποίο θεωρείται ότι οι μετασχηματισμοί 1, 2 και 3 λαμβάνουν και προετοιμάζουν τα δεδομένα εισόδου. Ο μετασχηματισμός 4 κάνει την κύρια δουλειά της δημιουργίας νέων δεδομένων και χαρακτηρίζεται ως κεντρικός μετασχηματισμός, ενώ οι μετασχηματισμοί 5, 6, 7 και 8 προετοιμάζουν τα δεδομένα εξόδου.

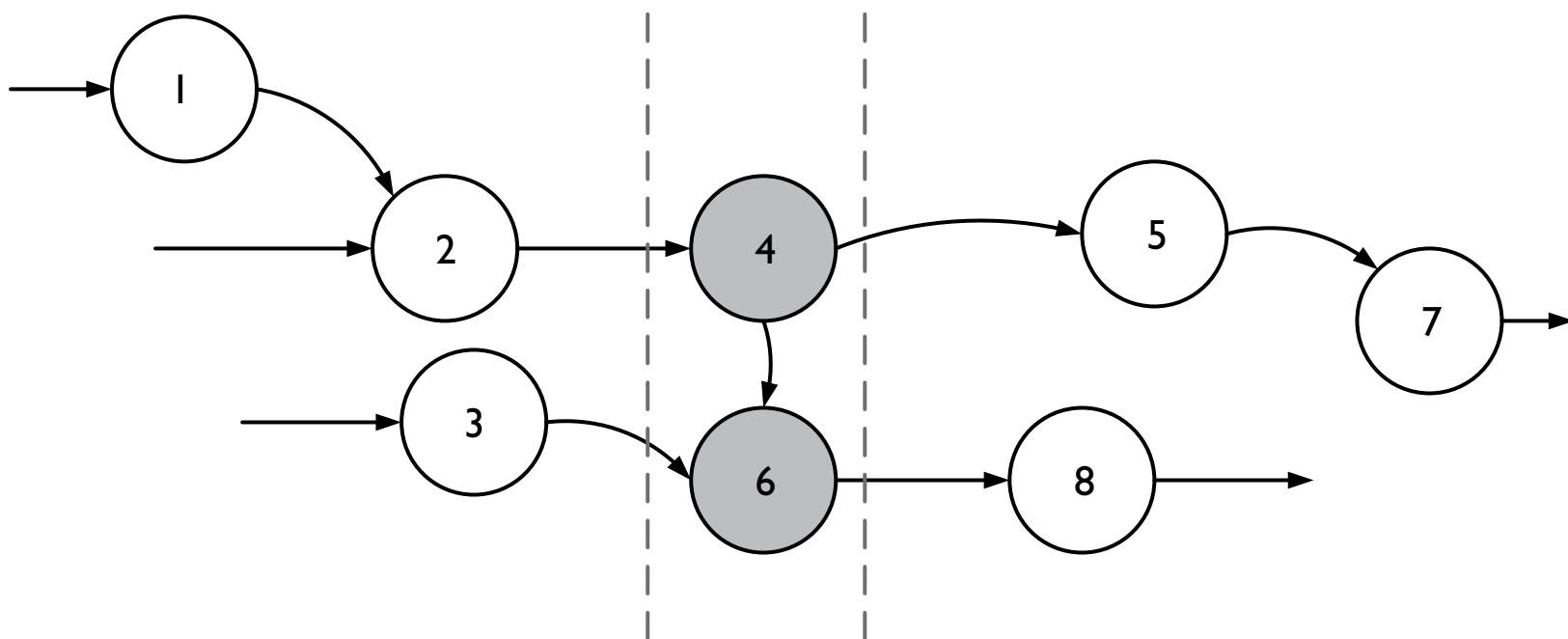
**Σχήμα 5.9** Ένα παράδειγμα εντοπισμού κεντρικού μετασχηματισμού.



Μια διαφορετική εκδοχή επί του ιδίου διαγράμματος ροής δεδομένων φαίνεται στο Σχήμα 5.10. Η ιδέα είναι ότι και ο μετασχηματισμός 6 ανήκει στον κεντρικό μετασχηματισμό. Καμία από τις δύο εκδοχές δεν μπορεί να χαρακτηριστεί ως καλύτερη από την άλλη χωρίς να είναι γνωστό το συγκεκριμένο πρόβλημα και ο ορισμός εκάστου των μετασχηματισμών του διαγράμματος ροής δεδομένων.

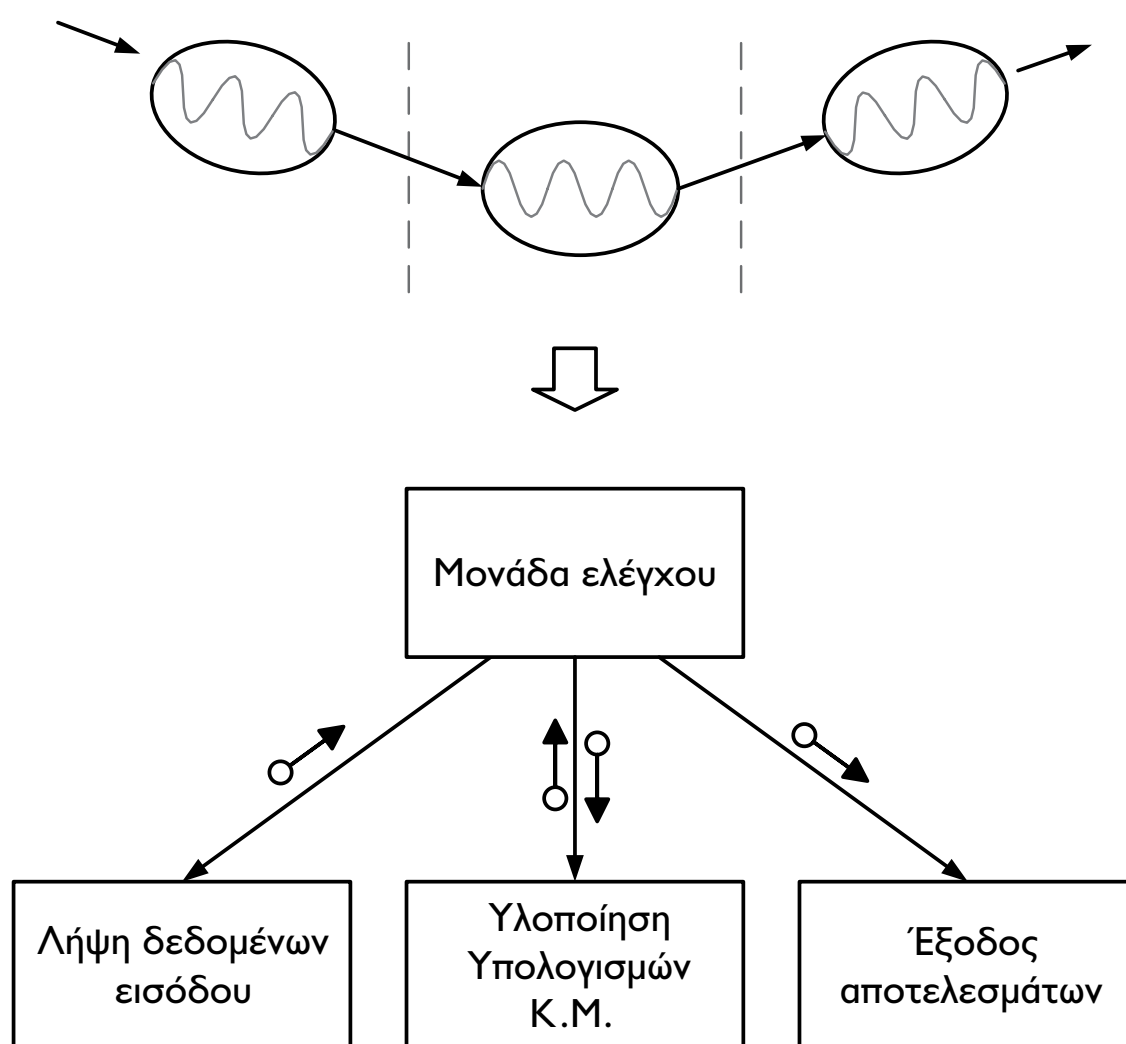


**Σχήμα 5.10** Μια εναλλακτική επιλογή του κεντρικού μετασχηματισμού του προηγούμενου σχήματος.



Ένας κεντρικός μετασχηματισμός απεικονίζεται σε διάγραμμα δομής, όπως φαίνεται στο Σχήμα 5.11. Η μονάδα ελέγχου εκτέλεσης καλεί τις μονάδες που απαιτείται για να λάβει τα δεδομένα εισόδου, τις μονάδες που απαιτείται να εκτελέσουν τον κεντρικό μετασχηματισμό και, τέλος, τις μονάδες στις οποίες θα διαθέσει τα δεδομένα εξόδου.

**Σχήμα 5.11** Απεικόνιση κεντρικού μετασχηματισμού σε διάγραμμα δομής.



Στο παραπάνω διάγραμμα μπορούν να υπάρχουν περισσότερες από μία μονάδες λήψης και εξόδου αποτελεσμάτων, ανάλογα με το πλήθος των εισερχόμενων και εξερχόμενων ροών στον κεντρικό μετασχηματισμό αντίστοιχα. Το σχέδιο που αντιστοιχεί στον κεντρικό μετασχηματισμό περιέχει τόσες μονάδες, όσοι και οι απλοί μετασχηματισμοί που περιλαμβάνονται στον κεντρικό μετασχηματισμό.

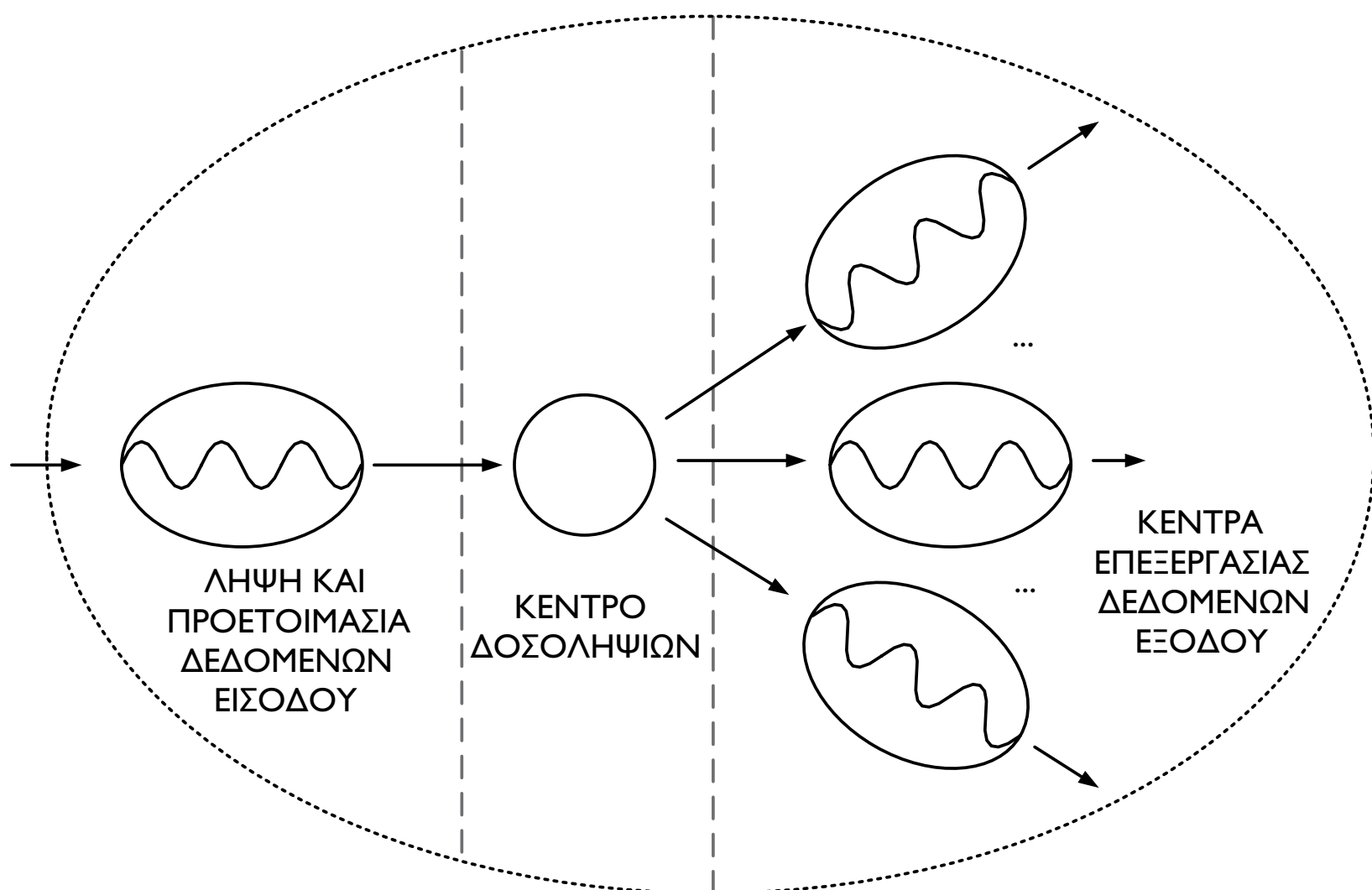
Μια δεύτερη χαρακτηριστική περιοχή ενός διαγράμματος ροής δεδομένων είναι το *κέντρο δοσοληψιών*.

### **Κέντρο δοσοληψιών:**

Ως κέντρο δοσοληψιών (transaction centre) ορίζεται ένας μετασχηματισμός του διαγράμματος ροής δεδομένων ο οποίος δέχεται κάποια δεδομένα εισόδου και παράγει ένα σύνολο δεδομένων εξόδου ανάλογα με την τιμή των δεδομένων εισόδου.

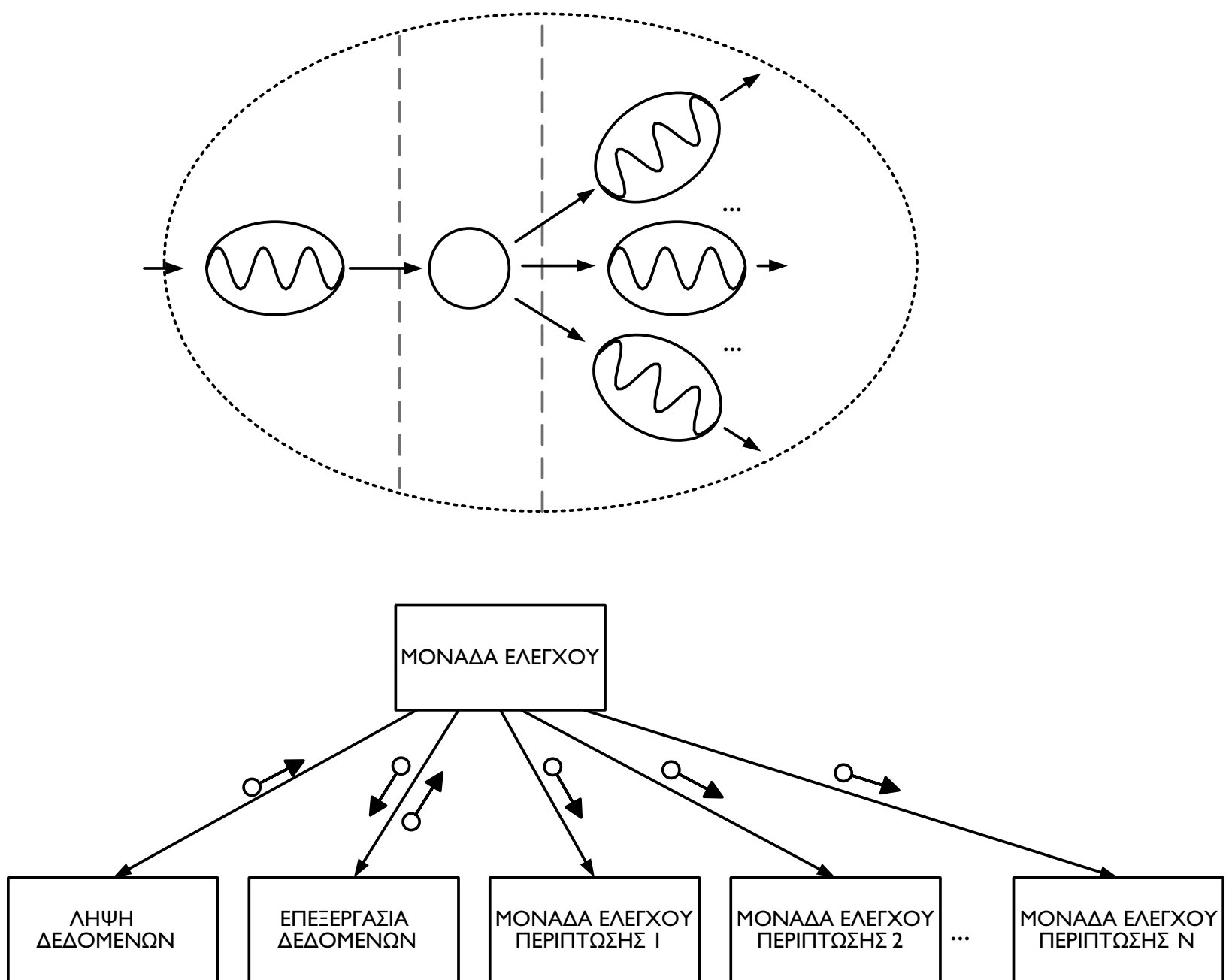
Αξίζει να σημειωθεί ότι ενώ ένας κεντρικός μετασχηματισμός μπορεί να περιλαμβάνει περισσότερους του ενός απλούς μετασχηματισμούς του διαγράμματος ροής δεδομένων, ένα κέντρο δοσοληψιών περιλαμβάνει μόνο έναν. Ένα κέντρο δοσοληψιών φαίνεται στο Σχήμα 5.12.

**Σχήμα 5.12** Η έννοια του κέντρου δοσοληψιών.



Χαρακτηριστική περίπτωση κέντρου δοσοληψιών είναι ένας μετασχηματισμός ελέγχου ροής προγράμματος όπου, ανάλογα με την επιλογή του χρήστη, η ροή μεταφέρεται σε διαφορετικούς μετασχηματισμούς, όπως η περίπτωση ενός μενού. Ένα κέντρο δοσοληψιών απεικονίζεται σε διάγραμμα δομής, όπως στο Σχήμα 5.13. Η μονάδα ελέγχου του κέντρου δοσοληψιών καλεί δύο μονάδες για τη λήψη και επεξεργασία των δεδομένων εισόδου αντίστοιχα, καθώς και τόσες μονάδες όσες είναι οι περιπτώσεις των δεδομένων εξόδου για τη μεταφορά του ελέγχου.

**Σχήμα 5.13** Απεικόνιση κέντρου δοσοληψιών σε διάγραμμα δομής.



### 5.4.2. Βήματα κατασκευής διαγραμμάτων δομής

Η μετάβαση από το διάγραμμα ροής δεδομένων στο διάγραμμα δομής γίνεται με διαδοχική επανάληψη κάποιων βημάτων, μέχρι να έχουν προσδιοριστεί μονάδες για όλους τους μετασχηματισμούς που περιέχονται στα διαγράμματα ροής δεδομένων της εφαρμογής. Τα βήματα αυτά είναι:

1. **Εντοπισμός κεντρικού μετασχηματισμού.** Για κάθε τμήμα του διαγράμματος ροής δεδομένων εντοπίζεται ο κεντρικός μετασχηματισμός και διακρίνονται οι μετασχηματισμοί εισόδου και εξόδου σε αυτόν.
2. **Απεικόνιση του κεντρικού μετασχηματισμού σε διάγραμμα δομής.** Δημιουργείται ένα επίπεδο του διαγράμματος δομής που αντιστοιχεί στον κεντρικό μετασχηματισμό.
3. **Παραγοντοποίηση (factoring).** Για το αριστερό και το δεξί τμήμα του κεντρικού μετασχηματισμού (είσοδοι και έξοδοι) δημιουργούνται τα διαγράμματα δομής που αντιστοιχούν στους μετασχηματισμούς που περιέχονται σε αυτά. Κάθε τέτοιος μετασχηματισμός απεικονίζεται σε μία μονάδα ελέγχου και σε δύο άλλες μονάδες. Από αυτές, η πρώτη λαμβάνει τα δεδομένα εισόδου, ενώ η δεύτερη πραγματοποιεί τη μετατροπή. Η μονάδα ελέγχου διαθέτει τα δεδομένα στο παραπάνω επίπεδο. Κατά την παραγοντοποίηση ενδεχομένως να αναγνωριστούν και άλλοι κεντρικοί μετασχηματισμοί ή κέντρα δοσοληψιών, τα οποία αντιμετωπίζονται όπως περιγράφηκε. Η διαδικασία επαναλαμβάνεται μέχρις ότου φτάσουμε στις πηγές και τους αποδέκτες των δεδομένων, δηλαδή το χρήστη, τα εξωτερικά συστήματα, τις συσκευές ή τα αρχεία.
4. **Συνένωση.** Η τελευταία εργασία που πρέπει να γίνει είναι αυτή της συνένωσης. Για τις περιπτώσεις που τα δεδομένα δεν λαμβάνονται από εξωτερική πηγή ή δεν καταλήγουν σε εξωτερικό αποδέκτη, τότε η μονάδα που τα εισάγει στον κεντρικό μετασχηματισμό αντικαθίσταται από τη μονάδα ελέγχου του μετασχηματισμού που τα παρέχει.

Κατά τη διαδικασία αυτή ενδεχομένως να διαπιστωθεί ότι είναι χρήσιμο να πραγματοποιηθούν τροποποιήσεις στα διαγράμματα ροής δεδομένων, γεγονός που συνιστά οπισθοδρόμηση στη διαδικασία ανάπτυξης λογισμικού, η οποία όμως είναι χρήσιμο να πραγματοποιηθεί. Ευθύς αμέσως θα δοθεί

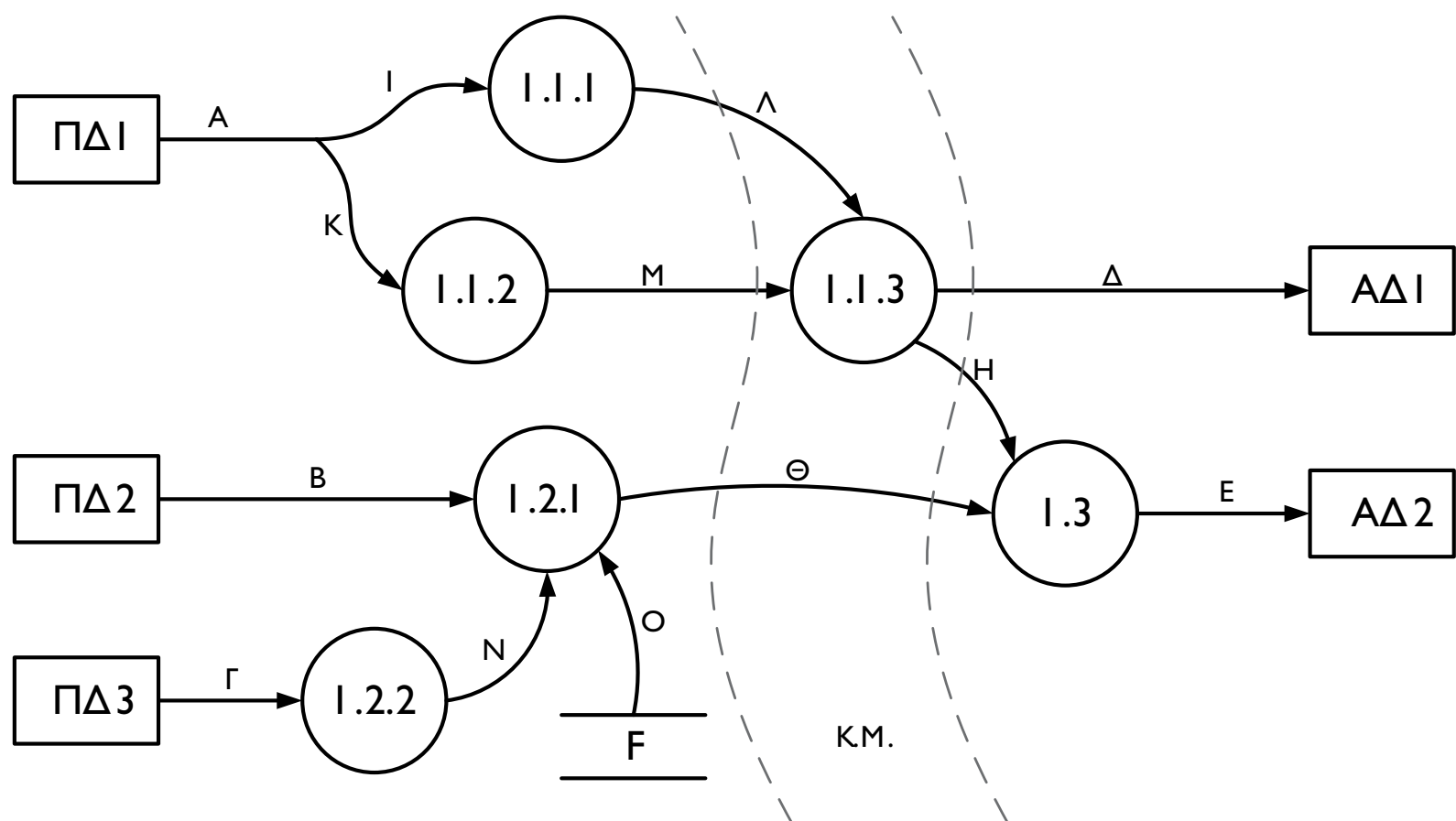


ένα παράδειγμα εφαρμογής των παραπάνω βημάτων σε ένα διάγραμμα ροής δεδομένων από το προηγούμενο κεφάλαιο.

### **Παράδειγμα I/Κεφάλαιο 5**

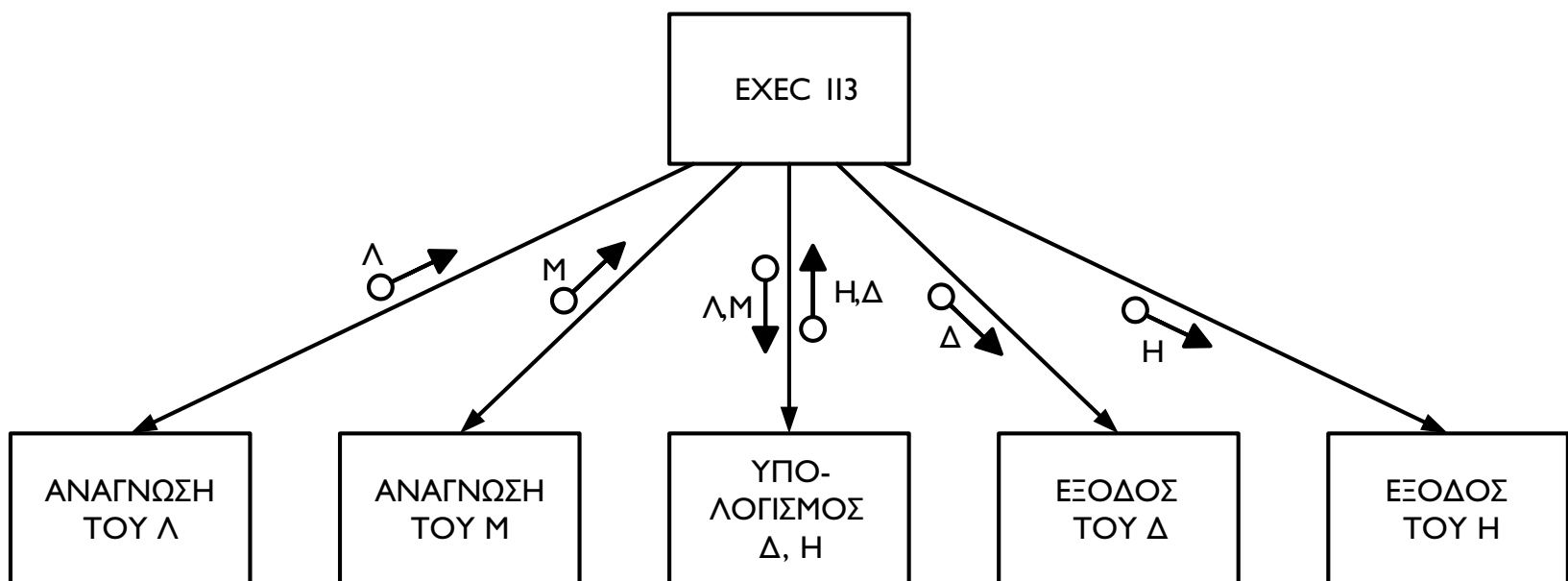
Στο Σχήμα 5.14 φαίνεται το διάγραμμα ροής δεδομένων του Σχήματος 4.9 χωρίς την ανάλυση του μετασχηματισμού I.3 στο τρίτο επίπεδο για λόγους απλότητας. Επιλέγουμε το μετασχηματισμό I.I.3 ως τον κεντρικό μετασχηματισμό (βήμα I) και κατασκευάζουμε το διάγραμμα δομής στο πρώτο επίπεδο.

**Σχήμα 5.14** Ένα παράδειγμα ορισμού κεντρικού μετασχηματισμού.



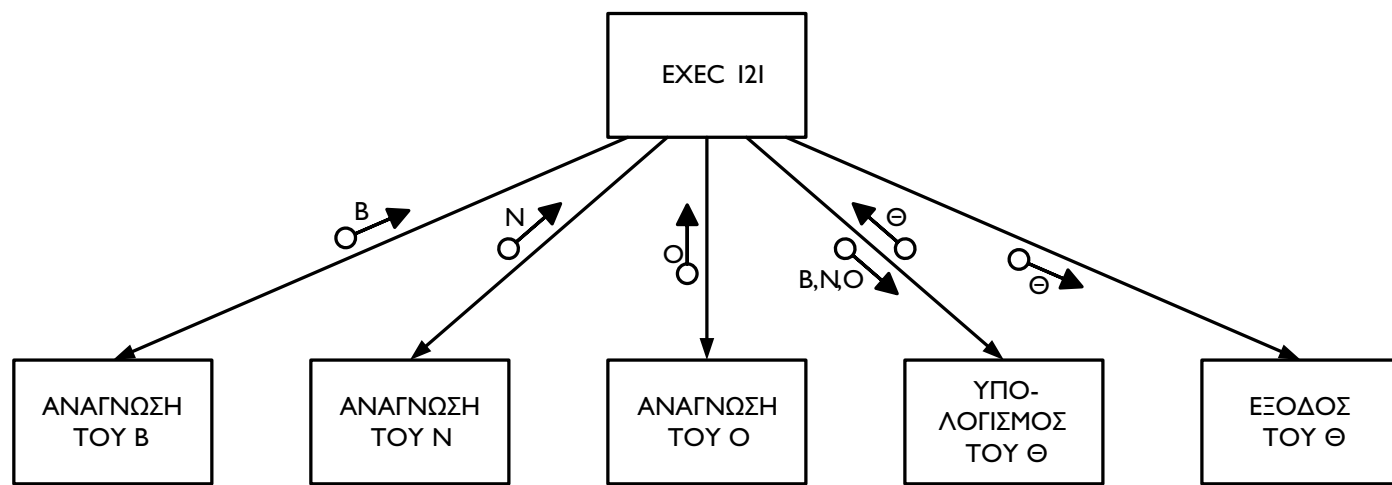
Με βάση την επιλογή αυτή κατασκευάζουμε το διάγραμμα δομής που φαίνεται στο Σχήμα 5.15 (εφαρμόζοντας το βήμα 2 που περιγράφηκε). Στον κεντρικό μετασχηματισμό εισέρχονται δύο ροές δεδομένων, οι  $\Lambda$  και  $M$ . Αυτές αντιστοιχούν στις μονάδες «ανάγνωση του  $\Lambda$ » και «ανάγνωση του  $M$ ». Τα δεδομένα αυτά περνάνε στη μονάδα «υπολογισμός  $\Delta, H$ », τα οποία είναι τα δεδομένα εξόδου από τον κεντρικό μετασχηματισμό. Αυτά διοχετεύονται προς δύο μονάδες, τις «έξοδος του  $\Delta$ » και «έξοδος του  $H$ » αντίστοιχα.

**Σχήμα 5.15** Το διάγραμμα δομής που αντιστοιχεί στο Σχήμα 5.14.

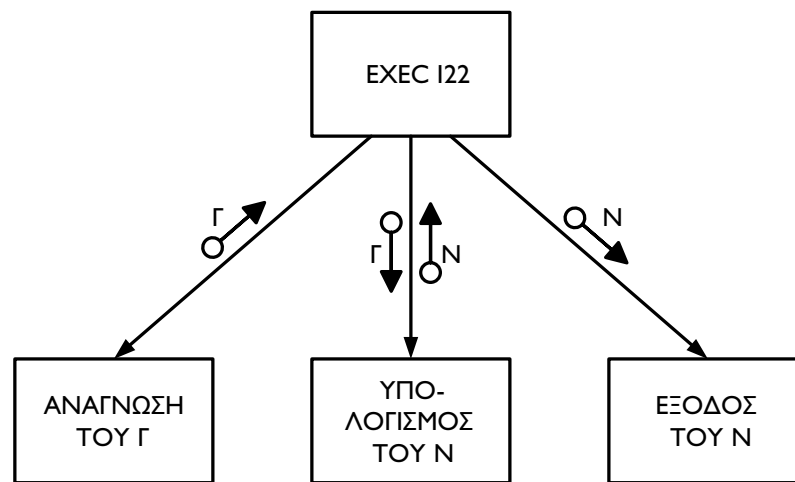


Η ροή Δ καταλήγει σε αποδέκτη δεδομένων (χρήστης) και δεν αναλύεται παραπάνω. Η ροή Η μεταφέρεται στη μονάδα I.3, η οποία τροφοδοτείται επίσης και από τη μονάδα I.2.1. Με τη σειρά της, η I.2.1 τροφοδοτείται με τη ροή Ν από τη μονάδα I.2.2. Ακολουθώντας εφαρμόζουμε τα αναφερόμενα στο βήμα 3, δηλαδή αναπτύσσουμε διαγράμματα δομής για κάθε μετασχηματισμό που προσφέρει δεδομένα εισόδου και αποδέχεται δεδομένα εξόδου προς/από τον κεντρικό μετασχηματισμό που έχουμε επιλέξει. Το αποτέλεσμα της διαδικασίας φαίνεται στο Σχήμα 5.16. Το διάγραμμα (α) αντιστοιχεί στο μετασχηματισμό I.2.1, το (β) στο μετασχηματισμό I.2.2, ενώ το (γ) στο μετασχηματισμό I.3.

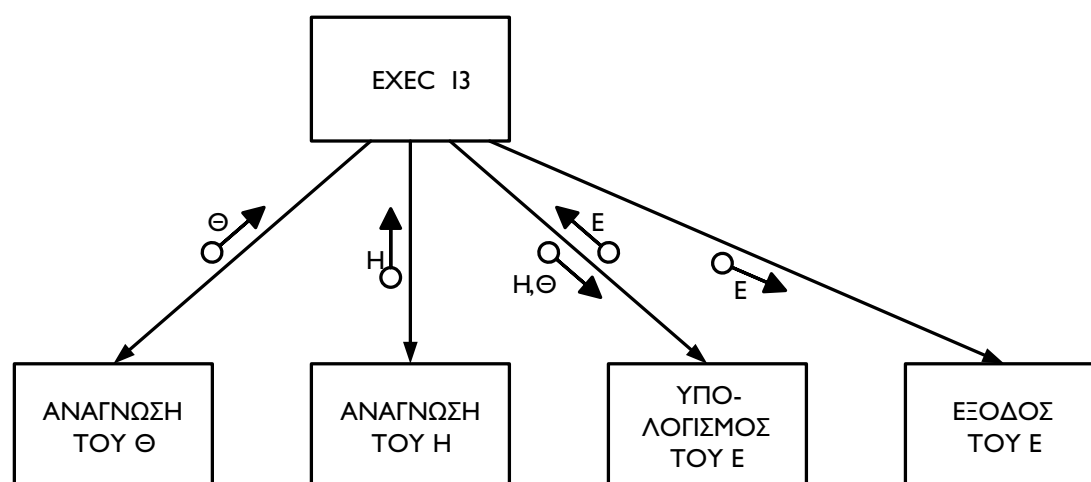
**Σχήμα 5.16** Παραγοντοποίηση των εισόδων στον κεντρικό μετασχηματισμό.



(α)



(β)

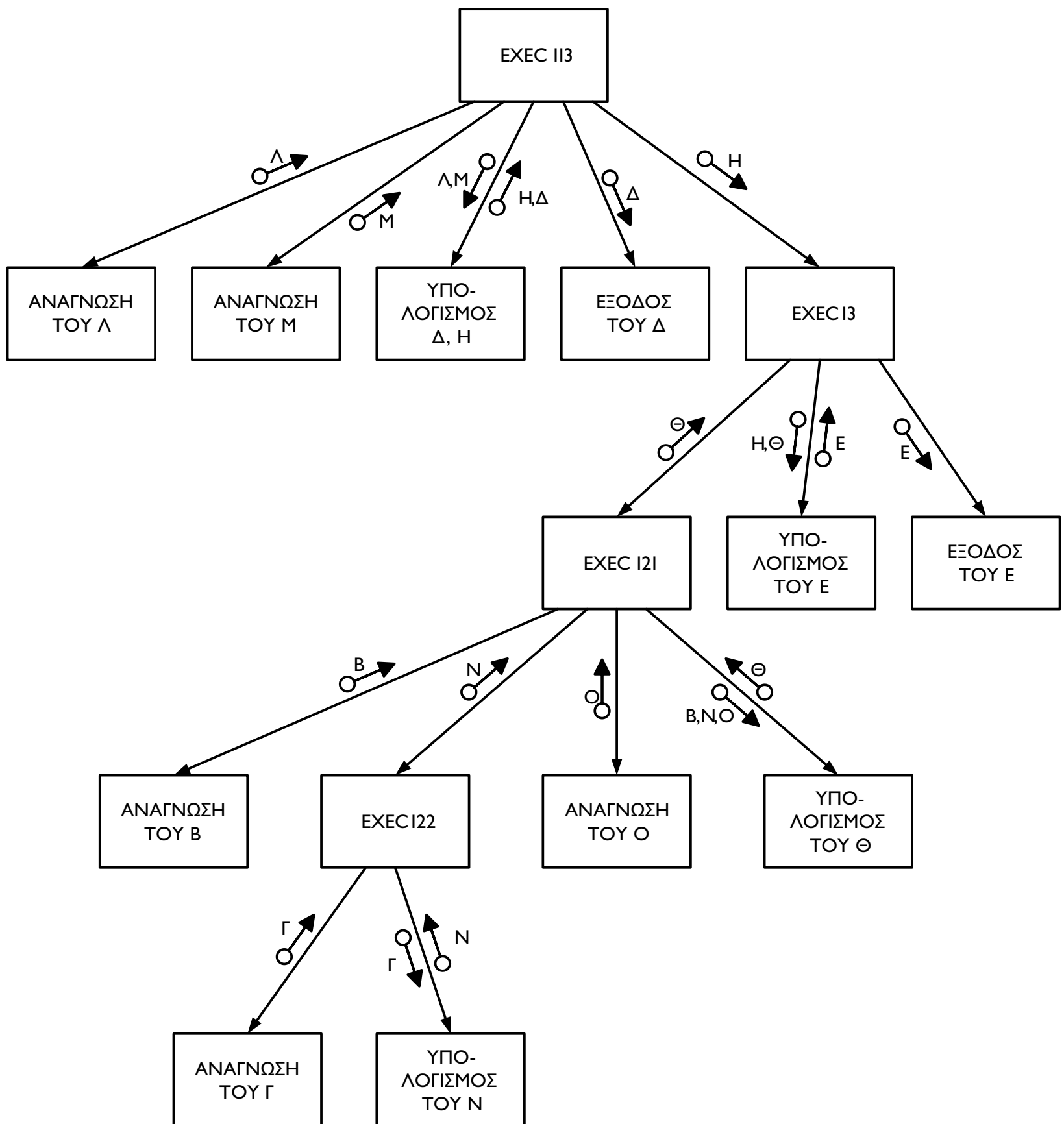


(γ)

Συνενώνουμε τα διαγράμματα δομής που έχουμε μέχρι το σημείο αυτό (βήμα 4) και καταλήγουμε στο διάγραμμα που φαίνεται στο Σχήμα 5.17. Η συνένωση βασίζεται στη μεταφορά των παραμέτρων και επαληθεύεται πρακτικά ως εξής: όταν μια μονάδα A ενός διαγράμματος δομής υψηλότερου επιπέδου διαβάζει ένα δεδομένο και το διαθέτει στη μονάδα ελέγχου της, ενώ μια μονάδα B ενός διαγράμματος δομής χαμηλότερου επιπέδου είναι μονάδα υποδοχής του ίδιου δεδομένου που έχει παραχθεί πριν, τότε η μονάδα A και η μονάδα ελέγχου της B ταυτίζονται. Με αυτό τον τρόπο πετυχαίνεται η συνένωση.

Μπορείτε να επαληθεύσετε αυτή την απλή διαδικασία θέτοντας στη θέση των A και B τις μονάδες «ανάγνωση του N» και «έξοδος του N» που περιέχονται στο Σχήμα 5.16.

**Σχήμα 5.17** Συνένωση διαγραμμάτων δομής.





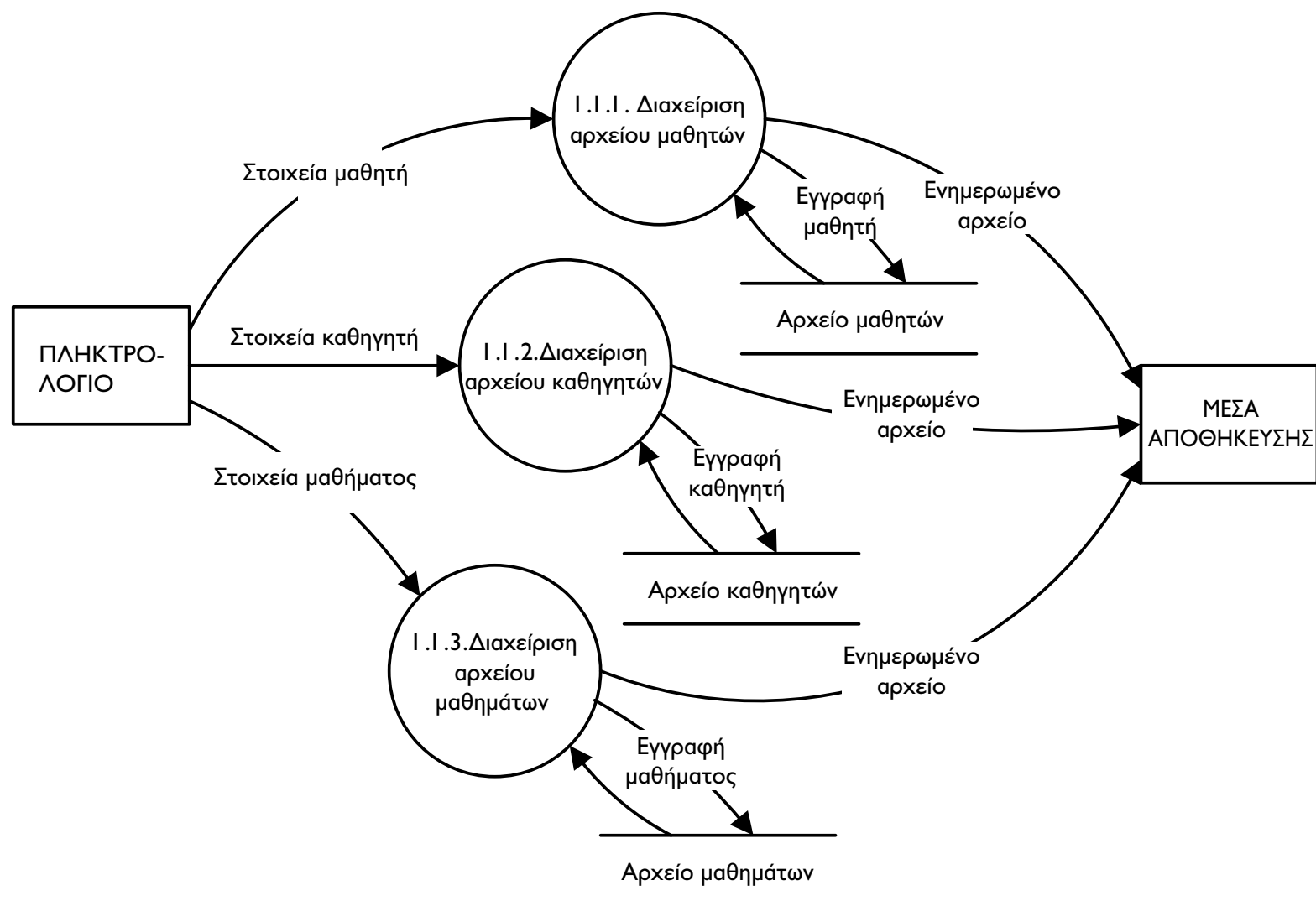
### Άσκηση 3/Κεφάλαιο 5

**Συμπληρώστε το διάγραμμα που φαίνεται στο Σχήμα 5.17 παραγοντοποιώντας τους μετασχηματισμούς I.I.1 και I.I.2 οι οποίοι περιέχονται στο Σχήμα 5.14.**

### Μελέτη περίπτωσης/Κεφάλαιο 5

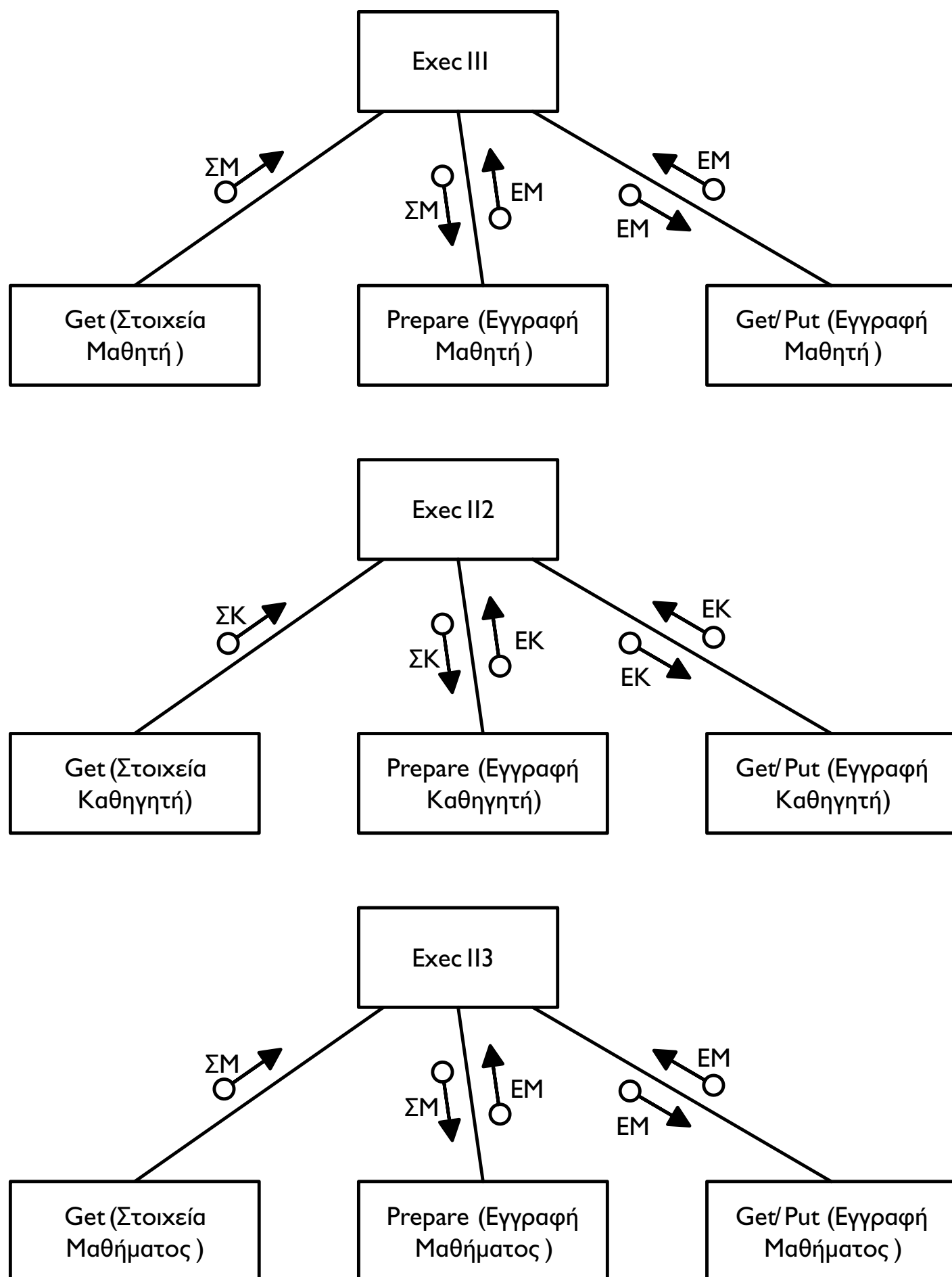
Ακολουθώντας θα επανέλθουμε στη μελέτη περίπτωσης που παρουσιάσαμε στο Κεφάλαιο 4 και θα κατασκευάσουμε τα διαγράμματα δομής της εφαρμογής λογισμικού «Επίκουρος». Στο Σχήμα 5.18 φαίνεται το πρώτο τμήμα του διαγράμματος ροής δεδομένων που εικονίζεται στο Σχήμα 4.11. Το διάγραμμα αυτό περιλαμβάνει τρεις απλούς μετασχηματισμούς, καθένας εκ των οποίων είναι κεντρικός μετασχηματισμός, διότι για κανέναν από αυτούς δεν υπάρχουν προηγούμενοι ή επόμενοι μετασχηματισμοί οι οποίοι μπορούν να θεωρηθούν ότι προσάγουν ή καταναλώνουν δεδομένα.

**Σχήμα 5.18** Τμήμα του διαγράμματος ροής δεδομένων της εφαρμογής λογισμικού «Επίκουρος», η οποία παρουσιάστηκε στο Κεφάλαιο 4.



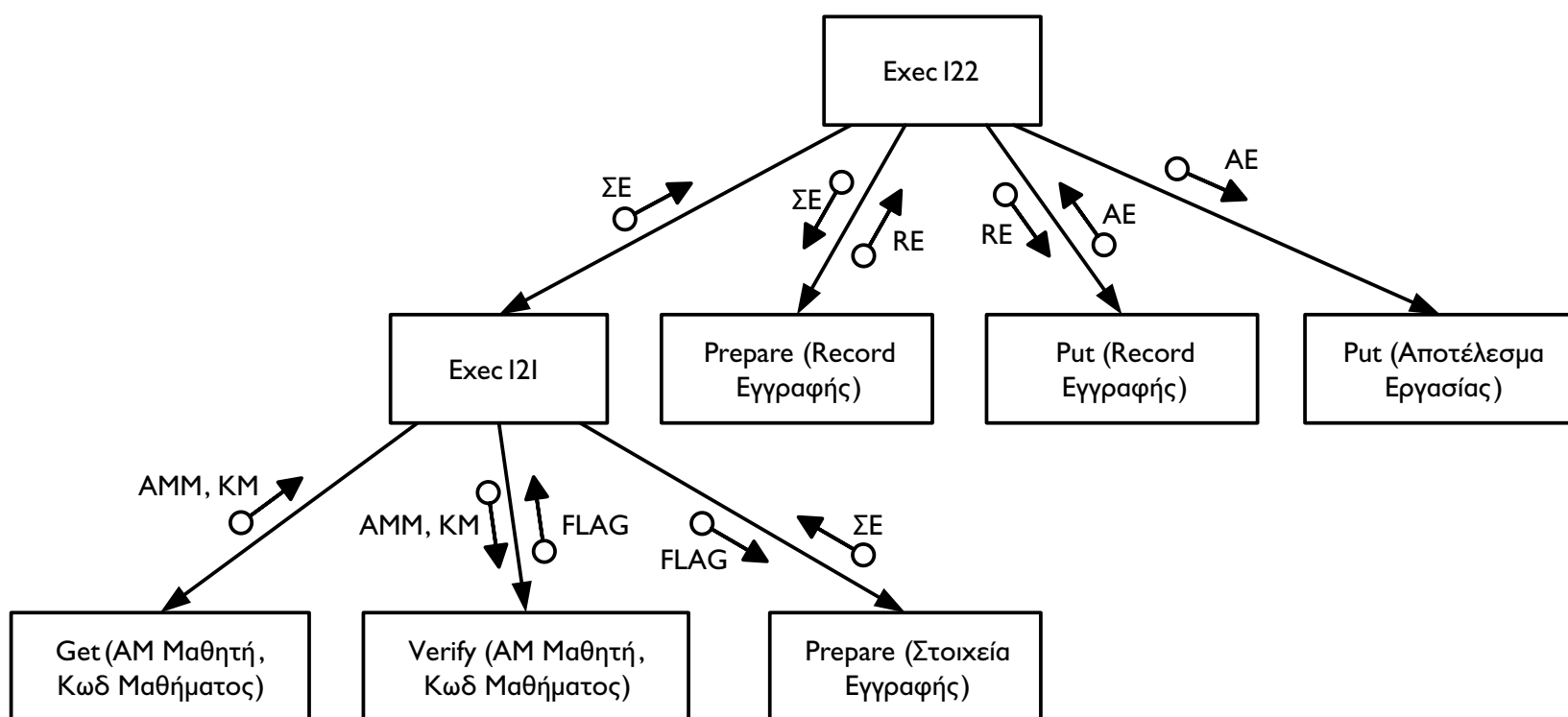
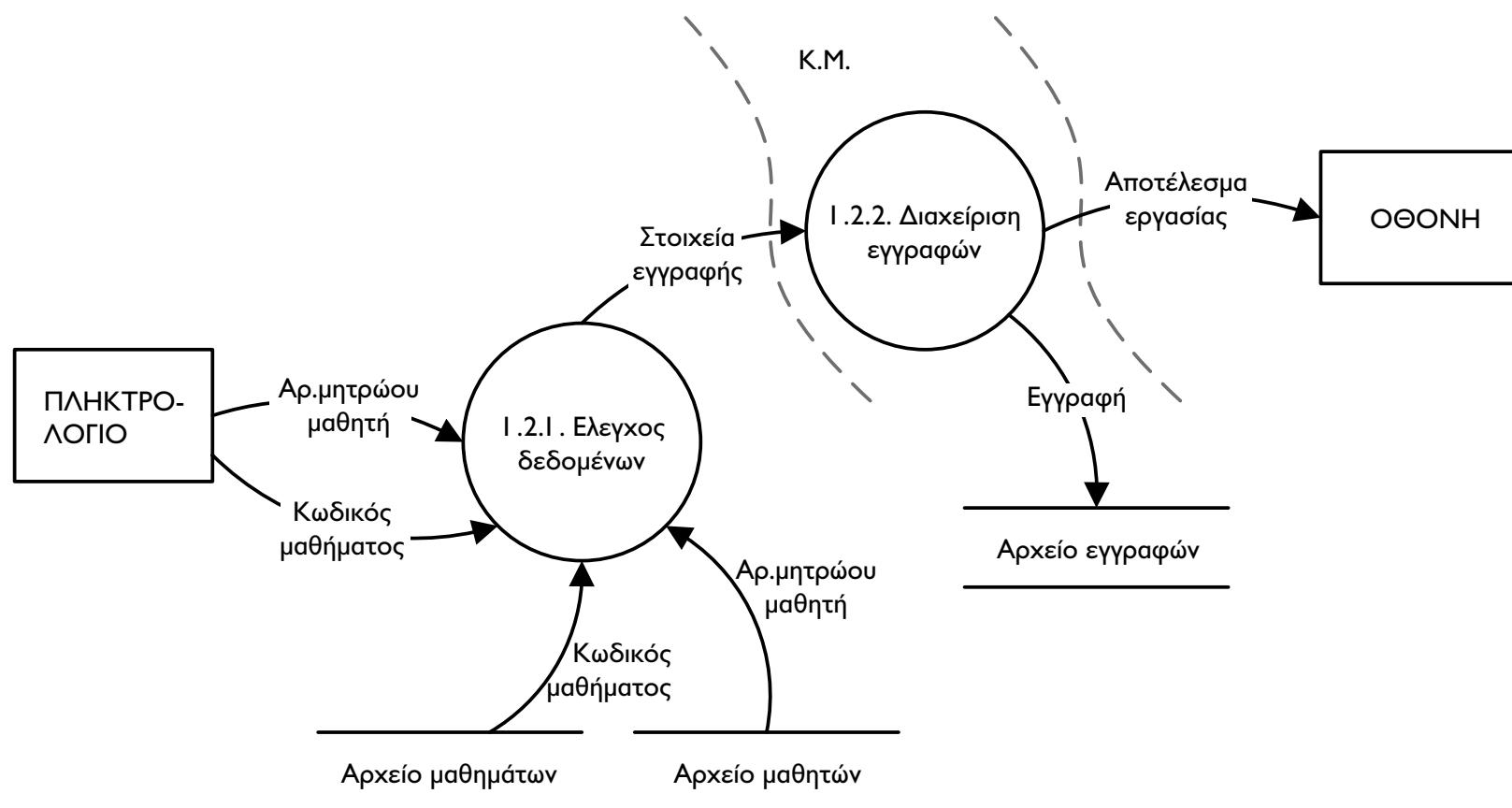
Τα διαγράμματα δομής που αντιστοιχούν στους μετασχηματισμούς αυτούς φαίνονται ακολούθως στο Σχήμα 5.19.

**Σχήμα 5.19** Τα διαγράμματα δομής που αντιστοιχούν στο Σχήμα 5.18.



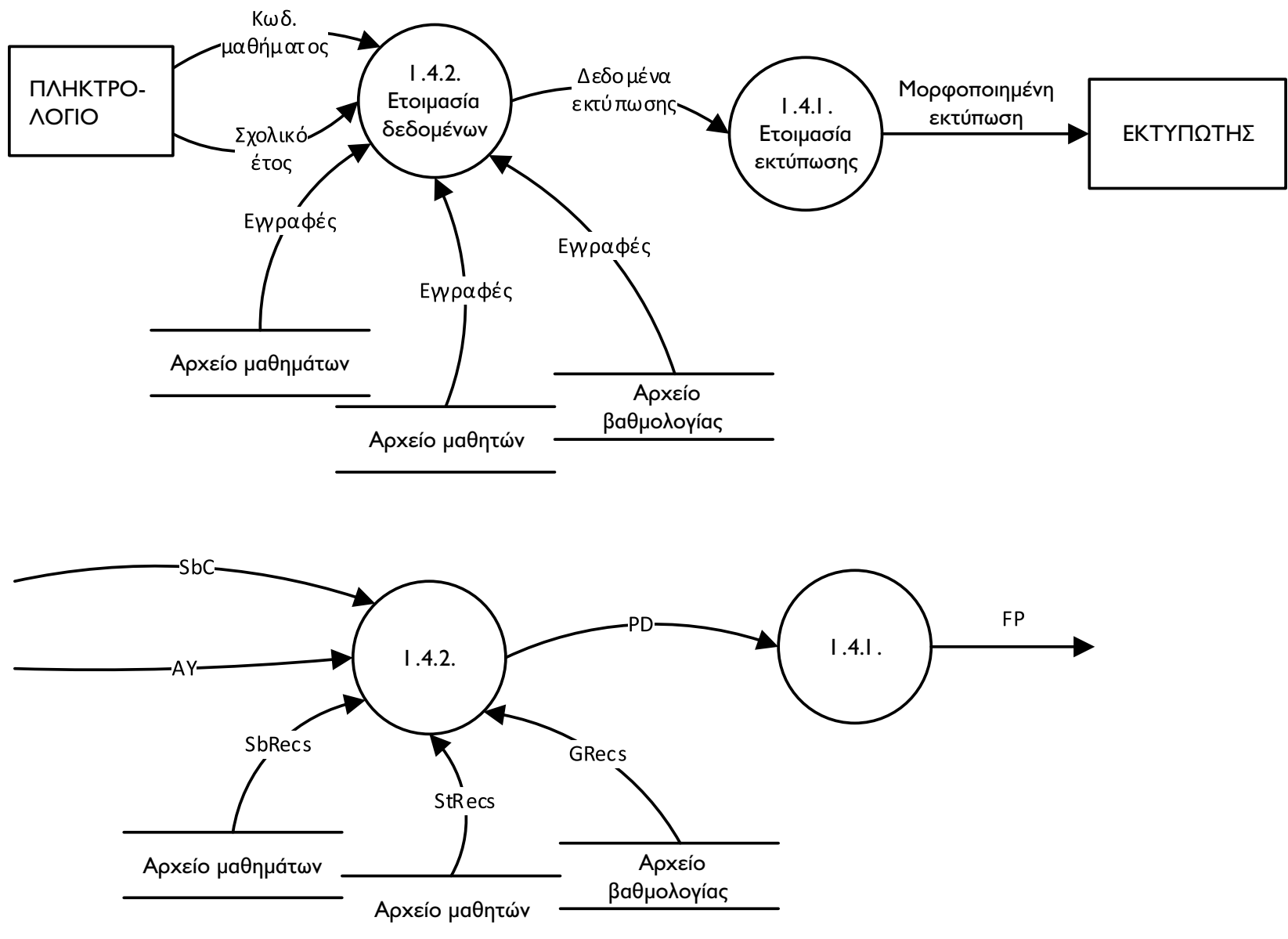
Το δεύτερο μέρος του διαγράμματος ροής δεδομένων του Σχήματος 4.11 φαίνεται ακολούθως στο Σχήμα 5.20, μαζί με το αντίστοιχο διάγραμμα δομής προγράμματος. Ως κεντρικός μετασχηματισμός επιλέγεται ο μετασχηματισμός I.2.2, διότι ο μετασχηματισμός I.2.1 ετοιμάζει τα δεδομένα, ο I.2.2 κάνει την κύρια εργασία και τα αποτελέσματα καταναλώνονται από τον χρήστη και το αρχείο. Στο διάγραμμα δομής φαίνεται και η παραγοντοποίηση του μετασχηματισμού I.2.1. Να σημειωθεί ότι η χρήση συντμήσεων και ρημάτων στην αγγλική γίνεται μόνο για οικονομία χώρου.

**Σχήμα 5.20** Επιλογή κεντρικού μετασχηματισμού μαζί με το αντίστοιχο διάγραμμα δομής προγράμματος.



Συνεχίζοντας, παραθέτουμε το δεύτερο μέρος του διαγράμματος ροής δεδομένων του Σχήματος 4.12. Από τη μελέτη του διαγράμματος προέκυψε ανάγκη τροποποίησής του, σύμφωνα με τα αναφερόμενα στην υποενότητα 5.4.2. Τα δεδομένα «κωδικός μαθήματος» και «σχολικό έτος» δεν είναι απαραίτητο να τροφοδοτούνται πρώτα στο μετασχηματισμό I.4.1 και μέσω αυτού στον I.4.2. Αντ' αυτού, μπορούν να τροφοδοτούνται κατευθείαν στον I.4.2. Η τροποποίηση αυτή φαίνεται στο Σχήμα 5.21 που ακολουθεί, μαζί με μια απλοποιημένη ως προς τα ονόματα των μετασχηματισμών και των δεδομένων εκδοχή του διαγράμματος ροής δεδομένων.

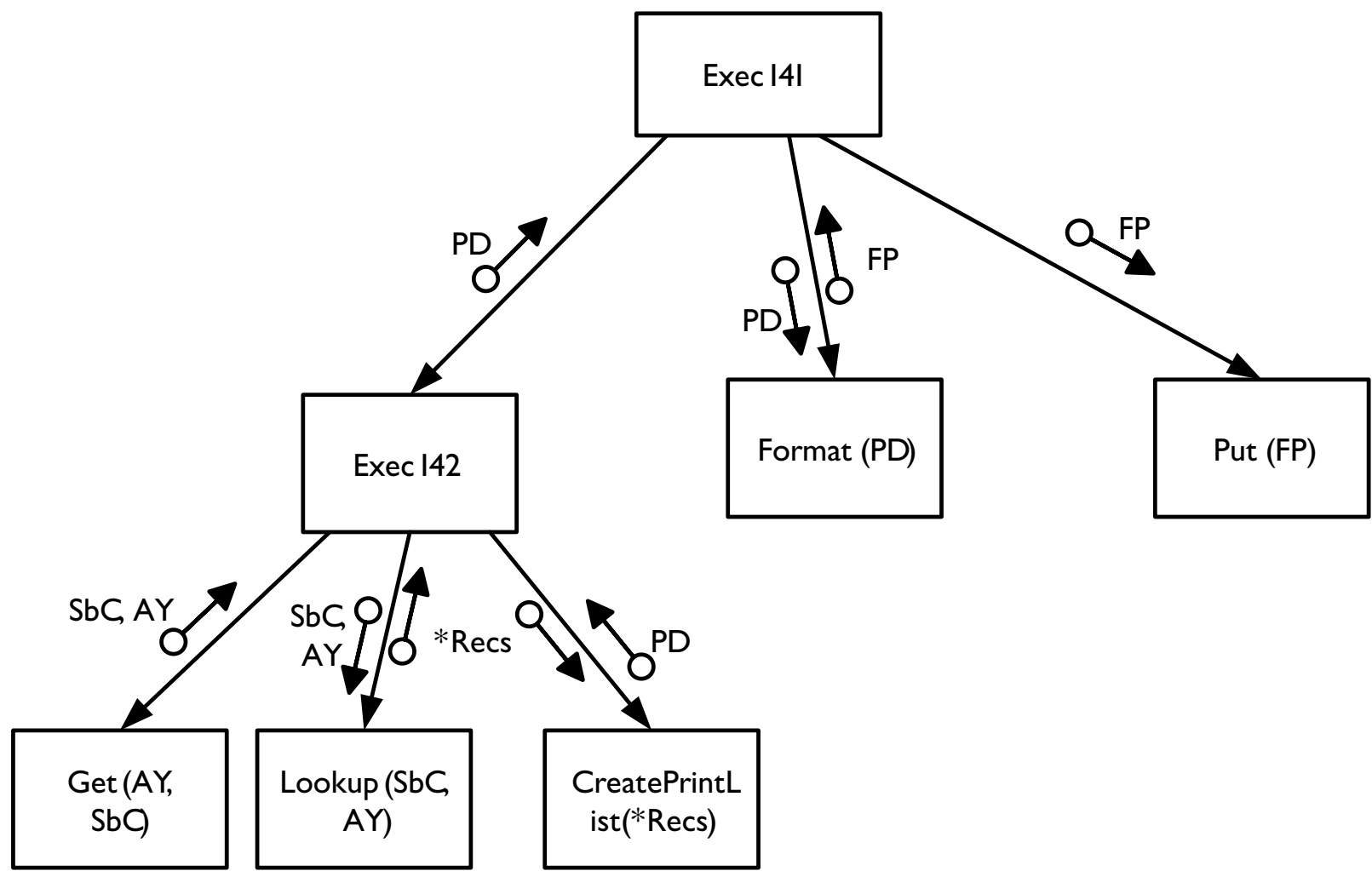
**Σχήμα 5.21** Τροποποιημένο διάγραμμα ροής δεδομένων του σχήματος 4.12, μαζί με μια εκδοχή με σύντομα ονόματα μετασχηματισμών και μεταβλητών.





Το ζητούμενο διάγραμμα δομής προγράμματος δίνεται ακολούθως στο Σχήμα 5.22.

**Σχήμα 5.22** Το διάγραμμα δομής προγράμματος που αντιστοιχεί στο Σχήμα 5.21.



## Άσκηση 4/Κεφάλαιο 5

Κατασκευάστε το διάγραμμα δομής προγράμματος για τους μετασχηματισμούς I.3.1 και I.3.2 της εφαρμογής λογισμικού «Επίκουρος» (Σχήμα 4.12).

### ΕΝΟΤΗΤΑ 5.5. ΛΕΠΤΟΜΕΡΗΣ ΣΧΕΔΙΑΣΗ ΜΟΝΑΔΩΝ

Όταν έχουμε διαθέσιμο το αρχιτεκτονικό σχέδιο, είναι δυνατή η λεπτομερής σχεδίαση των μονάδων λογισμικού που περιλαμβάνονται σε αυτό. Κατά τη λεπτομερή σχεδίαση θα προσδιοριστεί η εσωτερική δομή κάθε μονάδας, δηλαδή θα δοθεί μια περιγραφή του πηγαίου κώδικα. Μέχρι το σημείο αυτό είναι γνωστή μόνο η ονομασία κάθε μονάδας και οι παράμετροι εισόδου και εξόδου αυτής. Στοιχεία που σχετίζονται με την περιγραφή της συμπεριφοράς της είναι διαθέσιμα από τη φάση της προδιαγραφής των απαιτήσεων. Με το υλικό αυτό ο σχεδιαστής λογισμικού καλείται να κατασκευάσει το λεπτομερές σχέδιο.

Σε περίπτωση που το διάγραμμα δομής προγράμματος είναι επαρκώς λεπτομερές, καθεμία μονάδα που διαγράμματος αντιστοιχεί σε μία μονάδα κώδικα. Αν όμως αυτό δεν ισχύει, είναι ανάγκη κάθε μονάδα του διαγράμματος δομής να απεικονιστεί, ενδεχομένως, σε περισσότερες από μία μονάδες λογισμικού. Η εργασία αυτή γίνεται αναλύοντας κάθε πρόβλημα σχεδίασης σε διαδοχικά βήματα. Σε κάθε βήμα δημιουργούμε ένα σύνολο από μικρότερα προβλήματα σχεδίασης, καθένα εκ των οποίων επιλύουμε και πάλι αναλύοντάς το κ.ο.κ., μέχρις ότου φτάσουμε σε απλές δομικές μονάδες λογισμικού, όπως οι διαδικασίες και οι συναρτήσεις. Η αντιμετώπιση αυτή ονομάζεται «εκλέπτυνση σε διαδοχικά βήματα» (stepwise refinement) ή «συναρτησιακή αποσύνθεση» (functional decomposition).

Υπάρχουν αρκετοί τρόποι για να περιγράφονται τα αποτελέσματα της λεπτομερούς σχεδίασης. Ο επικρατέστερος είναι με χρήση μιας γλώσσας

σχεδίασης προγράμματος (PDL: Program Description Language). Μια γλώσσα σχεδίασης προγράμματος μοιάζει με γλώσσα προγραμματισμού χωρίς να έχει την αυστηρότητα στη σύνταξη και τη γραμματική που χαρακτηρίζει τις γλώσσες προγραμματισμού. Σκοπός της είναι να παρέχει μια εικόνα του λογισμικού η οποία να μπορεί να υλοποιηθεί εύκολα σε κάποια γλώσσα προγραμματισμού. Οι γλώσσες σχεδίασης προγράμματος περιέχουν τις βασικές δομές ελέγχου που απαντώνται στις γλώσσες προγραμματισμού. Μια απλή τέτοια γλώσσα, που ομοιάζει στην Pascal και στην οποία θα βασιστούν τα παραδείγματα που θα ακολουθήσουν, περιγράφεται στο Σχήμα 5.23. Οι γλώσσες σχεδίασης προγράμματος αναφέρονται συχνά και ως ψευδοκώδικας.

**Σχήμα 5.23** Μια απλή γλώσσα περιγραφής προγράμματος.

Απλές εκφράσεις	Επαναληπτική εκτέλεση
/*σχόλιο */ μεταβλητή := τιμή        /* ανάθεση */ φραστική περιγραφή ενέργειας + - * / ^        /* μαθηματικές εκφράσεις */	<b>FOR</b> μτβλ <b>FROM</b> τιμή1 <b>TO</b> τιμή2 <b>STEP</b> τιμή3 <b>DO</b> (ενέργειες) <b>END_FOR</b>
Εκτέλεση με επιλογή περίπτωσης	Εκτέλεση υπό συνθήκη
<b>CASE</b> έκφραση <b>OF</b> (τιμή 1) : (ενέργειες) (τιμή 2) : (ενέργειες) ... (τιμή N) : (ενέργειες) <b>OTHERWISE</b> (εντολές αν η έκφραση έχει άλλη τιμή) <b>END_CASE</b>	<b>IF</b> συνθήκη <b>THEN</b> (ενέργειες αν η συνθήκη είναι αληθής ) <b>ELSE</b> (εντολές αν η συνθήκη είναι ψευδής ) <b>END_IF</b>
Επαναληπτική εκτέλεση με συνθήκη	Επαναληπτική εκτέλεση με συνθήκη
<b>REPEAT</b> (ενέργειες) <b>UNTIL</b> συνθήκη	<b>WHILE</b> συνθήκη <b>DO</b> (ενέργειες) <b>END_WHILE</b>
Ορισμός διαδικασιών	Ορισμός συναρτήσεων
<b>PROCEDURE</b> όνομα (παράμετρος: <b>IN</b> / <b>OUT</b> , ...) <b>GLOBAL VAR</b> όνομα1 , όνομα2, ... <b>LOCAL VAR</b> όνομα1 , όνομα2, ... ... (ενέργειες) ... <b>CALL</b> όνομα_διαδικασίας (παράμ 1 , παράμ 2, ...) ... (ενέργειες) ... <b>END_PROCEDURE</b>	<b>FUNCTION</b> όνομα_συνάρτησης (παράμετρος , ...) <b>GLOBAL VAR</b> όνομα1 , όνομα2, ... <b>LOCAL VAR</b> όνομα1 , όνομα2, ... ... (ενέργειες) ... όνομα_συνάρτησης := τιμή ... (ενέργειες) ... <b>END_FUNCTION</b>

Προκειμένου να κατασκευαστεί το ομοίωμα αυτό (λεπτομερές σχέδιο), ο σχεδιαστής έρχεται αντιμέτωπος με αρκετά ζητήματα, όπως τα ακόλουθα:

- Ποιο είναι το καλύτερο από τα σχέδια τα οποία μπορεί να συλλάβει;
- Ποιος είναι ο πιο πρόσφορος τρόπος για την περιγραφή του λεπτομερούς σχεδίου μονάδων;
- Πώς σχετίζεται το σχέδιο με τη γλώσσα προγραμματισμού και το περιβάλλον υλοποίησης γενικότερα;
- Πώς μπορεί ένα λεπτομερές σχέδιο μονάδων να διατηρείται επίκαιρο;

Οι απαντήσεις στα ερωτήματα αυτά δεν είναι εύκολες και εξαρτώνται από πλήθος παραμέτρων και υποκειμενικών προσεγγίσεων. Στο Κεφάλαιο 6 αντιμετωπίζεται το πρόβλημα της συγγραφής πηγαίου κώδικα και αναφέρονται οι διαφορετικές πλευρές του προβλήματος κατασκευής του «καλύτερου» πηγαίου κώδικα. Η χρησιμότητα της ύπαρξης λεπτομερούς σχεδίου με τη βοήθεια ψευδοκώδικα έχει αμφισβητηθεί. Το βασικότερο επιχείρημα της αμφισβήτησης είναι ότι ο προγραμματιστής πρέπει να έχει την ελευθερία να αυτοσχεδιάσει. Αντίλογος εδώ μπορεί να εντοπιστεί σε ζητήματα όπως η ποιότητα, η αναγνωσιμότητα και η ευκολία συντήρησης του κώδικα, τα οποία θα μας απασχολήσουν στο επόμενο κεφάλαιο.

Σε πολλές περιπτώσεις δεν μπορεί να υποστηριχτεί με αξιώσεις η ανεξαρτησία του λεπτομερούς σχεδίου μονάδων από τη γλώσσα προγραμματισμού. Στην πράξη, ο σχεδιαστής έχει πάντα στο μυαλό του τη γλώσσα προγραμματισμού με την οποία θα υλοποιηθεί το σχέδιό του, και αυτό δεν είναι κακό εφόσον διευκολύνει τη μεταφορά του σχεδίου σε πηγαίο κώδικα. Το θέμα της ανεξαρτησίας σχεδίου λογισμικού και γλώσσας προγραμματισμού έχει αποτελέσει επί μακρόν αντικείμενο συζητήσεων μεταξύ των ερευνητών και των κατασκευαστών λογισμικού. Η λύση που φαίνεται να επικρατεί είναι η υπό παραδοχές και περιορισμούς ανεξαρτησία του σχεδίου από τον κώδικα με τη βοήθεια εργαλείων υποστήριξης της ανάπτυξης λογισμικού.

Στα εργαλεία βρίσκεται και η πιο προσγειωμένη απάντηση στο τελευταίο ερώτημα, αυτό της διατήρησης του σχεδίου του λογισμικού σε επίκαιρη

μορφή. Όπως έχει ήδη αναφερθεί, το λογισμικό δεν παραμένει στατικό και υπόκειται σε πολλές αλλαγές κατά τον κύκλο ζωής του είτε για τη διόρθωση σφαλμάτων είτε για τη μεταβολή των χαρακτηριστικών του. Αν και συνήθως οι αλλαγές πρέπει να γίνονται πρώτα στο σχέδιο και κατόπιν στον πηγαίο κώδικα, στην πράξη αυτό δεν συμβαίνει και οι αλλαγές συχνά πραγματοποιούνται κατευθείαν στον κώδικα. Έτσι, ένα σχέδιο που έχει κατασκευαστεί με το χέρι συχνά καταλήγει άχρηστο, εφόσον δεν ανταποκρίνεται στην πραγματικότητα. Κάτι τέτοιο είναι λιγότερο πιθανό να συμβεί όταν ο κατασκευαστής χρησιμοποιεί κάποιο εργαλείο το οποίο είτε διευκολύνει είτε υποχρεώνει τον κατασκευαστή να διατηρεί ενημερωμένο σχέδιο.

### **Παράδειγμα 2/Κεφάλαιο 5**

Θα κατασκευάσουμε το λεπτομερές σχέδιο των μονάδων λογισμικού ExecIII και GET\_ΣΜ που φαίνονται στο Σχήμα 5.19.

```

/*-----*/
PROCEDURE Exec111
/*-----*/
LOCAL VAR στοιχεία_μαθητή, εγγραφή_μαθητή
Αρχικοποίησε στοιχεία_μαθητή, εγγραφή_μαθητή

WHILE στοιχεία_μαθητή <> κενό DO

    CALL Get_ΣΜ(στοιχεία_μαθητή)
    IF στοιχεία_μαθητή <> κενό THEN
        CALL Prepare_ΣΜ(στοιχεία_μαθητή, εγγραφή_μαθητή)
        CALL Put_EM(εγγραφή_μαθητή)
    END_IF

END_WHILE

END_PROCEDURE

/*-----*/
PROCEDURE Get_ΣΜ(στοιχεία_μ: IN/OUT)
/*-----*/
Εμφάνισε φόρμα στην οθόνη
Διάβασε τα πεδία από το πληκτρολόγιο
IF ο χρήστης πάτησε ESC THEN
    Μηδένισε τις τιμές όλων των πεδίων της στοιχεία_μ
END_IF

END_PROCEDURE

```



Η διαδικασία αυτή διαβάζει από το πληκτρολόγιο στοιχεία μαθητών και ετοιμάζει μια εγγραφή του αρχείου μαθητών την οποία και αποθηκεύει. Η εργασία αυτή επαναλαμβάνεται μέχρις ότου τα στοιχεία μαθητή που θα δοθούν να είναι κενά. Είναι φανερό ότι, προκειμένου να καταλήξουμε σε άμεσα υλοποιήσιμη περιγραφή λεπτομερούς σχεδίου, είναι αναγκαίο να συνεχίσουμε εκλεπτύνοντας τις εργασίες «εμφάνισε φόρμα στην οθόνη» και «διάβασε τα πεδία από το πληκτρολόγιο», σύμφωνα με την αρχή της συναρτησιακής αποσύνθεσης.

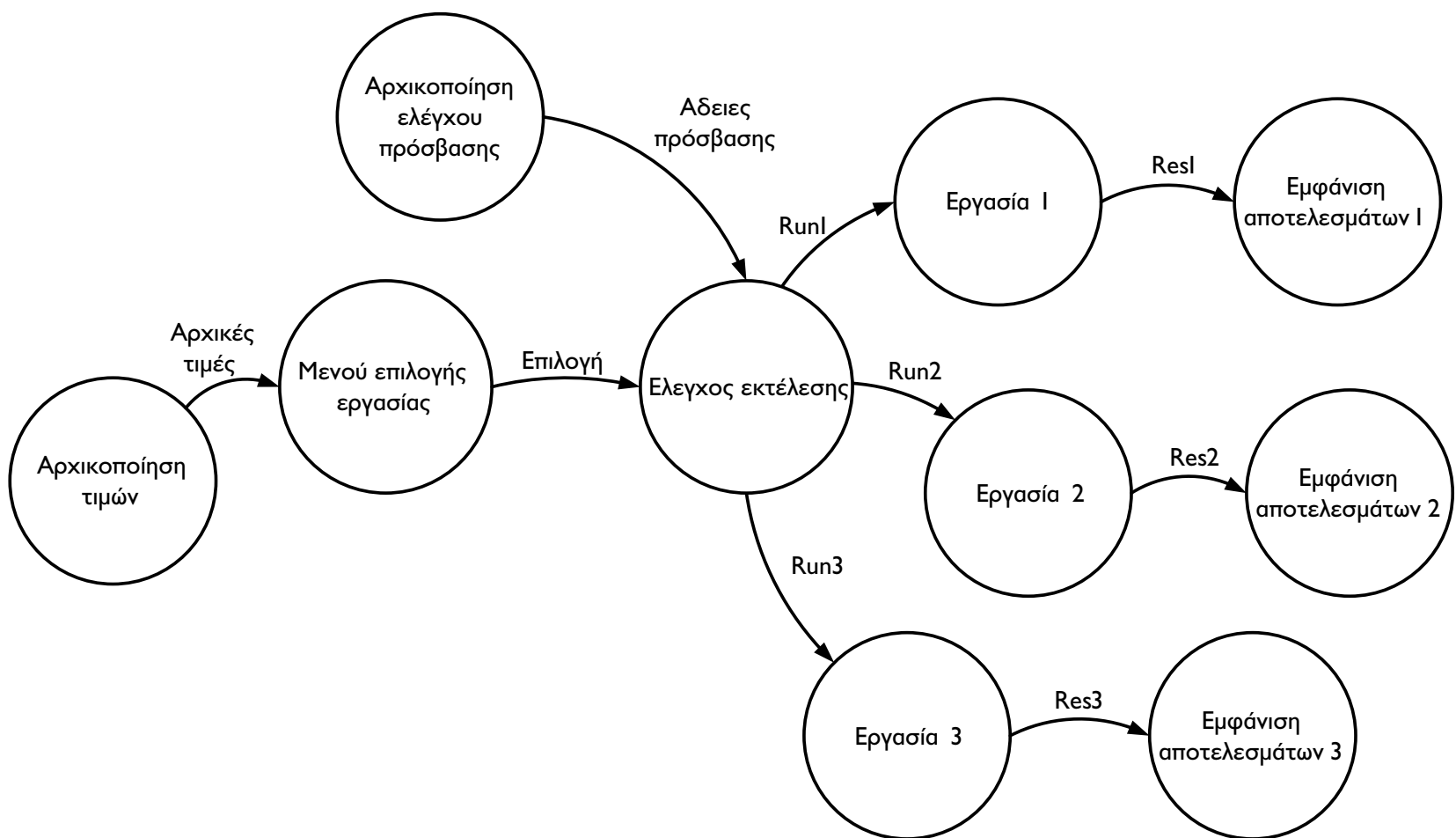
### **Δραστηριότητα 2/Κεφάλαιο 5**

Σε αναλογία με το παράδειγμα, κατασκευάστε το λεπτομερές σχέδιο των μονάδων ExecI4I και ExecI42 που φαίνονται στο Σχήμα 5.22.

### **Δραστηριότητα 3/Κεφάλαιο 5**

Στο διάγραμμα ροής δεδομένων που φαίνεται στο Σχήμα 5.24 που ακολουθεί, αναγνωρίστε ένα κέντρο δοσοληψιών και κατασκευάστε το διάγραμμα δομής προγράμματος και το λεπτομερές σχέδιο της μονάδας ελέγχου.

**Σχήμα 5.24** Ένα διάγραμμα ροής δεδομένων.



## ΕΝΟΤΗΤΑ 5.6. ΣΧΕΔΙΑΣΗ ΔΕΔΟΜΕΝΩΝ

Σκοπός της σχεδίασης δεδομένων είναι ο προσδιορισμός των δομών δεδομένων η ύπαρξη των οποίων εντοπίστηκε κατά τη φάση της προδιαγραφής των απαιτήσεων από το λογισμικό. Ο προσδιορισμός αυτός πρέπει να είναι επαρκής για την απεικόνιση των δεδομένων σε ένα περιβάλλον υλοποίησης. Η εργασία της σχεδίασης δεδομένων είναι στενά συνυφασμένη με την εργασία κατάστρωσης των υπόλοιπων τμημάτων του σχεδίου του λογισμικού στις οποίες αναφερθήκαμε. Συχνά, η εργασία σχεδίασης δεδομένων υποτιμάται από τους κατασκευαστές σε σχέση με την υπόλοιπη σχεδίαση.

Η σχεδίαση δεδομένων μπορεί να μελετηθεί αυτοτελώς σε σημαντικό βάθος και πλάτος. Χωρίς κάτι τέτοιο να αποτελεί αντικείμενο του παρόντος, παραθέτουμε τις εργασίες που πρέπει οπωσδήποτε να εκτελούνται κατά τη λεπτομερή σχεδίαση δεδομένων:

- Τροποποίηση του μοντέλου οντοτήτων – συσχετίσεων του οποίου την αρχική μορφή κατασκευάσαμε κατά τη συγγραφή των προδιαγραφών με σκοπό αυτό να περιέχει ακριβώς όση πληροφορία απαιτείται, κατανεμημένη σωστά μεταξύ των οντοτήτων. Η διαδικασία αυτή ονομάζεται «κανονικοποίηση».
- Καθορισμός των πεδίων που περιέχει κάθε πίνακας στο επίπεδο της φυσικής αποθήκευσης δεδομένων.
- Καθορισμός των εναλλακτικών μορφών εμφάνισης της οργάνωσης των δεδομένων πάνω από το φυσικό επίπεδο (ορισμός views).
- Βελτιστοποιήσεις με σκοπό τη βελτιστοποίηση των επιδόσεων και γενικά την επίτευξη κριτηρίων επάρκειας.

Η εργασία απεικόνισης οντοτήτων του πεδίου της εφαρμογής με τη βοήθεια ενός σχήματος δεδομένων είναι από τις πιο ενδιαφέρουσες, δημιουργικές και σημαντικές εργασίες κατά την ανάπτυξη του λογισμικού. Μερικές αρχές που είναι χρήσιμο να ακολουθούνται κατά τη διαδικασία αυτή είναι οι ακόλουθες:

- Η σχεδίαση δεδομένων δεν πρέπει να υποτιμάται. Η απόδοση χρόνου και σημασίας σε αυτή και η πειθαρχημένη εφαρμογή αρχών σχεδίασης ανάλογων με αυτές που χρησιμοποιούνται για την κατασκευή του αρχιτεκτονικού σχεδίου του λογισμικού ανταποδίδουν πάντα το χρόνο που καταναλώνουν.
- Θα πρέπει να εντοπιστούν όλες οι δομές δεδομένων πάνω στα οποία επιδρούν οι μονάδες λογισμικού. Μετά από τη σχεδίαση δεν θα πρέπει να υπάρχουν δεδομένα των οποίων ο προσδιορισμός να αφήνεται για το μέλλον.
- Το λεξικό δεδομένων, στο οποίο αναφερθήκαμε στο Κεφάλαιο 3, θα πρέπει να τηρείται ενημερωμένο.
- Οι αποφάσεις σχεδίασης δεδομένων που σχετίζονται με κατασκευαστικές λεπτομέρειες θα πρέπει να λαμβάνονται όσο αργότερα γίνεται. Η επιδίωξη αποφυγής της εξάρτησης των αποτελεσμάτων της σχεδίασης δεδομένων από το περιβάλλον υλοποίησης (όπως ένα σύστημα διαχείρισης βάσεων δεδομένων) συνήθως απαιτεί σημαντική προσπάθεια.
- Δεν είναι καλή ιδέα οποιαδήποτε μονάδα λογισμικού να μπορεί να διαβάσει και να μεταβάλει δεδομένα. Είναι σκόπιμο κατά τη σχεδίαση να καθορίζεται ποιες μονάδες λογισμικού επιτρέπεται να επιδρούν κατευθείαν στα δεδομένα. Με τον τρόπο αυτό μειώνονται οι πιθανότητες παρενεργειών στο λογισμικό καθώς πραγματοποιούνται μεταβολές στην εσωτερική δομή των δεδομένων.
- Θα πρέπει να διερευνάται η δυνατότητα χρήσης έτοιμων και δοκιμασμένων δομών δεδομένων από βιβλιοθήκες. Αν και η γενικευμένη χρήση βιβλιοθηκών συστατικών λογισμικού δεν έχει γίνει μέχρι σήμερα δυνατή, η εφαρμογή της ιδέας σε μικρή κλίμακα μέσα στην εμβέλεια ενός κατασκευαστή λογισμικού συχνά αποδίδει.

Το αποτέλεσμα της σχεδίασης δεδομένων είναι το τελικό διάγραμμα οντοτήτων – συσχετίσεων, το ενημερωμένο λεξικό δεδομένων και οι παράγραφοι 2.3, 3.3 και 6 του εγγράφου περιγραφής του σχεδίου του λογισμικού.

## ΕΝΟΤΗΤΑ 5.7. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ ΑΥΤΟΑΞΙΟΛΟΓΗΣΗΣ

### 5.7.1. Δραστηριότητες

#### Δραστηριότητα I/Κεφάλαιο 5

---

Στη δομημένη ανάλυση και σχεδίαση, το λογισμικό περιγράφεται ως μία ιεραρχία συστατικών η οποία επιδρά σε κάποια ανεξάρτητα από αυτή δεδομένα. Στην αντικειμενοστρεφή σχεδίαση, δεδομένα και ενεργά συστατικά λογισμικού συνυπάρχουν σε οντότητες που ονομάζονται «αντικείμενα». Κάθε αντικείμενο περιέχει ένα υποσύνολο δεδομένων και κάποια ενεργά συστατικά των οποίων η απόδοση έχει γίνει με κριτήριο το πεδίο ευθύνης του αντικειμένου. Η ουσιώδης διαφορά της δομημένης από την αντικειμενοστρεφή σχεδίαση είναι η ανεξαρτησία των δεδομένων από τα συστατικά λογισμικού που χαρακτηρίζει την πρώτη, ενώ δεν ισχύει στη δεύτερη.

Μπορείτε να αναπτύξετε πολύ περισσότερο τις θέσεις αυτές αν ανατρέξετε στη σχετική βιβλιογραφία. Είναι φυσιολογικό να μη συλλαμβάνετε την απάντηση αυτή με την πρώτη προσπάθεια.

## Δραστηριότητα 2/Κεφάλαιο 5

---

```
/*-----*/  
PROCEDURE Exec141  
/*-----*/  
LOCAL VAR Print_Data, Formatted_Printout  
  
Αρχικοποίησε Print_Data, Formatted_Printout  
  
CALL Exec142(Print_Data)  
  
WHILE Print_Data <> κενό DO  
    CALL Format(Print_Data, Formatted_Printout)  
    CALL Put(Formatted_Printout)  
    CALL Exec142(Print_Data)  
END_WHILE  
  
END_PROCEDURE
```

Η διαδικασία λαμβάνει τα προς εκτύπωση δεδομένα και, όσο αυτά δεν είναι κενά (που σημαίνει ότι ο χρήστης δεν ζήτησε τερματισμό της εργασίας), τα μορφοποιεί και τα στέλνει στη μονάδα εξόδου.

```
/*-----*/  
PROCEDURE Exec142(PrintList: IN/OUT)  
/*-----*/  
LOCAL VAR AcademicYear, SubjectCode, RecordList  
  
Αρχικοποίησε AcademicYear, SubjectCode, RecordList  
  
CALL Get_pref(AcademicYear, SubjectCode)  
CALL Lookup(AcademicYear, SubjectCode, RecordList)  
CALL CreatePrintList(RecordList, PrintList)  
  
END_PROCEDURE
```

Πρόκειται ουσιαστικά για μια μονάδα υλοποίησης ελέγχου ροής η οποία διαβάζει δεδομένα εισόδου, τα επιβεβαιώνει (Lookup) και τα στέλνει στη μονάδα που είναι υπεύθυνη για τη δημιουργία των δεδομένων της εκτύπωσης. Η περίπτωση όπου τα δεδομένα είναι κενά αντιμετωπίζεται στη μονάδα CreatePrintList.

Η χρήση του ψευδοκώδικα απαιτεί αρκετή εξοικείωση. Μην αποθαρρύνεστε αν δεν τη διαθέτετε από τώρα. Μετά από λίγο καιρό ενασχόλησης με το θέμα σε παραδείγματα και ασκήσεις, θα νιώθετε την απαιτούμενη εμπιστοσύνη. Αξίζει ένα «μπράβο» σε αυτούς που προσέγγισαν τη δική μας ή έδωσαν μία ακόμη καλύτερη απάντηση.

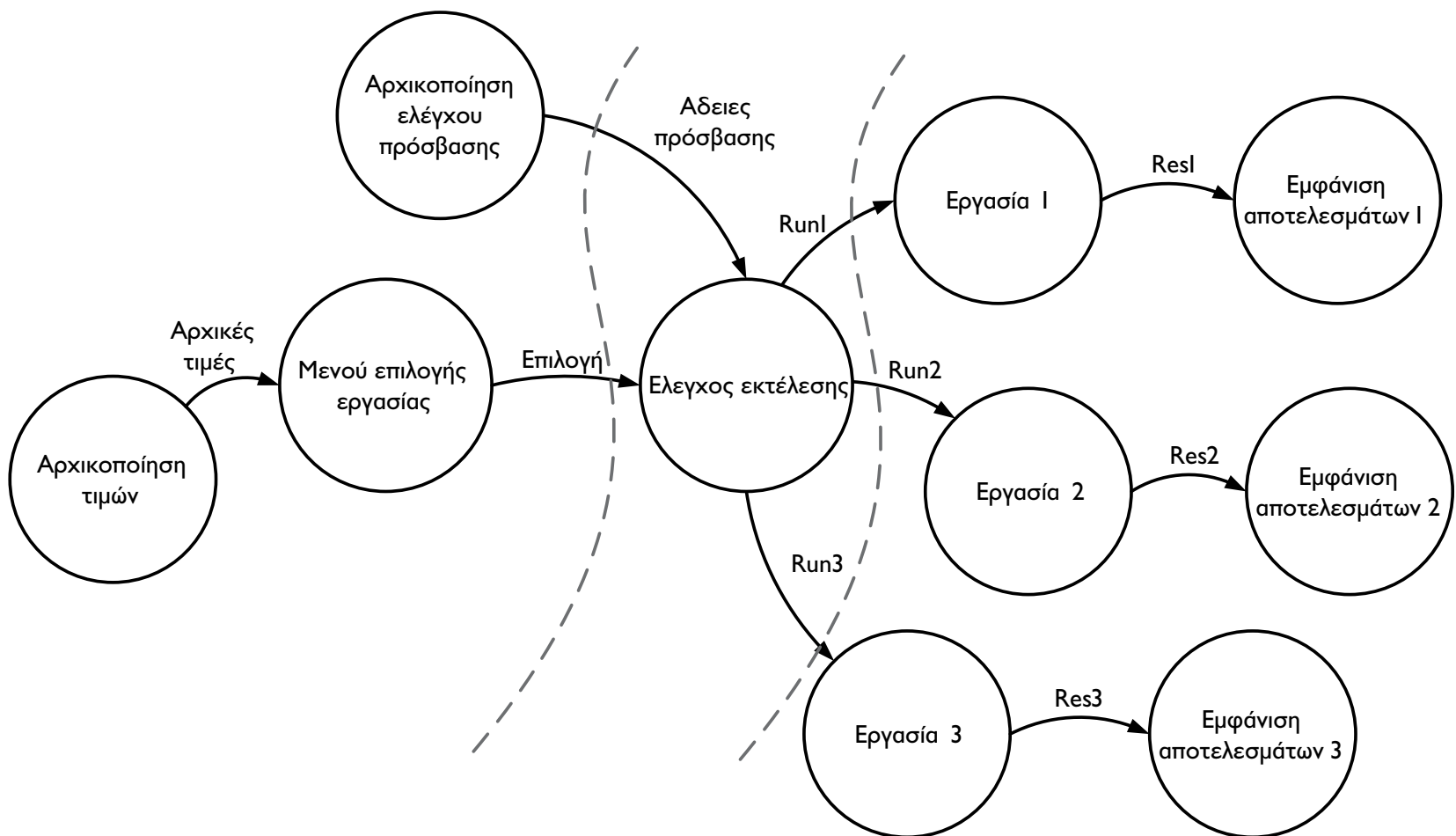
### Δραστηριότητα 3/Κεφάλαιο 5

---

Στο σχήμα της εκφώνησης αρχικά θα επιλεγεί το κέντρο δοσοληψιών. Σύμφωνα με όσα αναφέρονται στην Ενότητα 5.5, στο διάγραμμα ροής δεδομένων που δίνεται εμφανίζεται εικόνα ανάλογη με αυτή που φαίνεται στο Σχήμα 5.12 για τον μετασχηματισμό «έλεγχος εκτέλεσης». Οπότε αυτό είναι και το κέντρο δοσοληψιών.

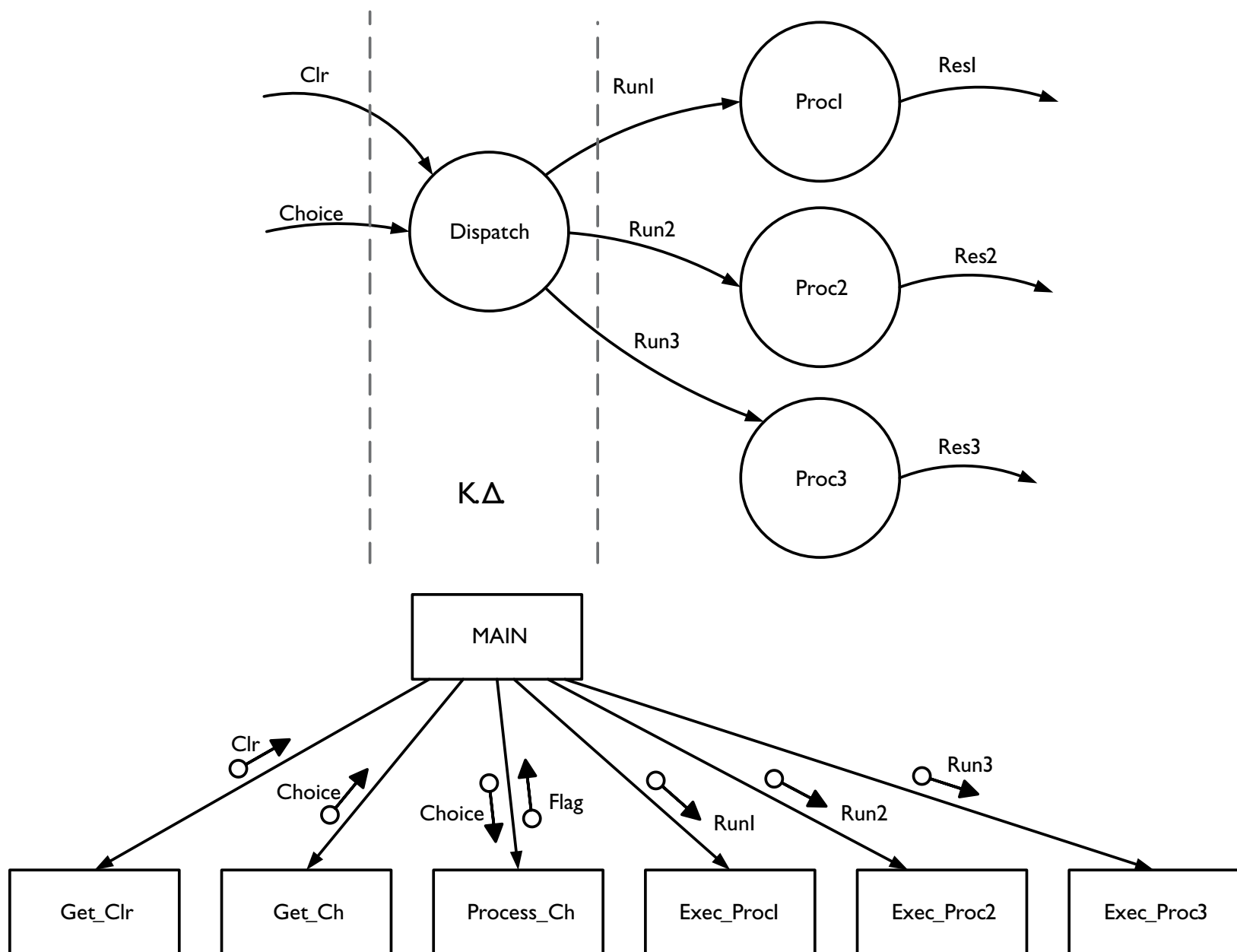


**Σχήμα 5.25** Επιλογή του κέντρου δοσοληψιών.



Για τη δημιουργία του ζητούμενου διαγράμματος δομής μάς απασχολούν μόνο οι μετασχηματισμοί που συνδέονται άμεσα με το κέντρο δοσοληψιών διαθέτοντας ή καταναλώνοντας δεδομένα, οι οποίοι και καταγράφονται με έντονα γράμματα στο Σχήμα 5.25. Στο Σχήμα 5.26 που ακολουθεί ξανασχεδιάζουμε το τμήμα που μας ενδιαφέρει από το Σχήμα 5.25, χρησιμοποιώντας ονόματα στην αγγλική για οικονομία χώρου. Ακολούθως φαίνεται το διάγραμμα δομής προγράμματος που αντιστοιχεί σε αυτό και το λεπτομερές σχέδιο μονάδων με τη μορφή ψευδοκώδικα.

**Σχήμα 5.26** Δραστηριότητα 3.



Δεν πρόκειται παρά για εφαρμογή όσων αναφέρθηκαν στην Ενότητα 5.5.I, όπου η επιλογή του κέντρου δοσοληψιών είναι μοναδική και η κατασκευή του διαγράμματος δομής γίνεται με άμεση εφαρμογή των κατευθύνσεων που δίνονται στο δεύτερο τμήμα της Ενότητας 5.5.I. Αν δεν τα καταφέρατε, μπορείτε να ανατρέξετε και πάλι στο τμήμα αυτό.

```
/*-----*/
PROCEDURE ExecMain
/*-----*/
LOCAL VAR Clr, Choice, Flag, Run1, Run2, Run3

Αρχικοποίησε Clr, Choice, Flag, Run1, Run2, Run3

CALL Get_Clr(Clr)
CALL Get_Choice(Choice)
CALL Process_Choice(Choice, Flag)

CASE Flag OF
    1 : CALL Exec_Proc1(Run1)
    2 : CALL Exec_Proc2(Run2)
    3 : CALL Exec_Proc3(Run3)
END_CASE

END_PROCEDURE
```

Στην εκδοχή αυτή τα δεδομένα εισόδου διαβάζονται μόνο μία φορά, καλείται η μονάδα επεξεργασίας (Process\_Choice) και κατόπιν η μονάδα που αντιστοιχεί στην επιλογή που έγινε. Μια εκδοχή, η οποία επαναλαμβάνει την ενέργεια επιλογής και εκτέλεσης εργασίας μέχρις ότου ο χρήστης να επιλέξει τον τερματισμό της εκτέλεσης, είναι και η ακόλουθη:

```

/*-----*/
PROCEDURE ExecMain2
/*-----*/
LOCAL VAR Clr, Choice, Flag, Run1, Run2, Run3

Αρχικοποίησε Clr, Choice, Flag, Run1, Run2, Run3

CALL Get_Clr(Clr)
CALL Get_Choice(Choice)
CALL Process_Choice(Choice, Flag)

WHILE Flag<>κενό DO

    CASE Flag OF
        1 : CALL Exec_Proc1(Run1)
        2 : CALL Exec_Proc2(Run2)
        3 : CALL Exec_Proc3(Run3)
    END_CASE

    CALL Get_Clr(Clr)
    CALL Get_Choice(Choice)
    CALL Process_Choice(Choice, Flag)

END_WHILE

END_PROCEDURE

```

Είναι φανερό ότι στις προδιαγραφές του λογισμικού πρέπει να περιέχεται το ποια είναι η εκδοχή που θα πρέπει να υλοποιηθεί. Είναι επίσης φανερό ότι είναι πολύ εύκολο αυτό να μη συμβαίνει και οι προδιαγραφές να

είναι ελλιπείς, οπότε η απόφαση θα πρέπει να ληφθεί από τον μηχανικό λογισμικού.

### 5.7.2. Ασκήσεις αυτοαξιολόγησης

#### Άσκηση 1/Κεφάλαιο 5

---

Η παραδοχή που θεωρείται ότι ισχύει είναι ότι κατά την έναρξη της σχεδίασης θεωρούνται γνωστές οι απαιτήσεις από το λογισμικό. Αυτό όμως ισχύει μόνο στιγμιαία, δηλαδή κατά την έναρξη της σχεδίασης. Συγχαρητήρια σε όσους έδωσαν αβίαστα την απάντηση. Στην πράξη, η εμπειρία με την ανάπτυξη έργων λογισμικού δυστυχώς επιβεβαιώνει τη θέση αυτή, η οποία συνήθως γίνεται συνειδητή μετά τη χρέωση του κόστους πραγματοποίησης μεταβολών στο λογισμικό για απαιτήσεις που τέθηκαν εκ των υστέρων.

Ο αναγνώστης παραπέμπεται και στη σχετική συζήτηση στο Κεφάλαιο Ι, όπου αναφέρεται ότι οι απαιτήσεις από το λογισμικό μπορεί να μεταβάλλονται ακόμη και κατά τη διάρκεια ανάπτυξης του λογισμικού. Οι μεταβολές αυτές προκαλούν οπισθοδρομήσεις και ταλανισμούς στη σχεδίαση, οι οποίες στοιχίζουν περισσότερο, όσο σε πιο προχωρημένο σημείο της σχεδίασης συμβαίνουν.

#### Άσκηση 2/Κεφάλαιο 5

---

Κατά την αρχιτεκτονική σχεδίαση μπορούν να συμπληρωθούν οι ενότητες:

- 2.1 Αποσύνθεση σε μονάδες
- 2.2 Αποσύνθεση σε ταυτόχρονες διεργασίες
- 3.1 Εξαρτήσεις μεταξύ μονάδων
- 3.2 Εξαρτήσεις μεταξύ διεργασιών

Κατά τη σχεδίαση διεπαφών συμπληρώνεται η ενότητα **4**.

Κατά τη λεπτομερή σχεδίαση μονάδων συμπληρώνεται η ενότητα **5**.

Κατά τη σχεδίαση δεδομένων συμπληρώνονται οι ενότητες:

- 2.3 Αποσύνθεση δεδομένων

### 3.3 Εξαρτήσεις μεταξύ δεδομένων

#### 4 Λεπτομερές σχέδιο δεδομένων

Η απάντηση απορρέει ως λογικό επακόλουθο όσων αναφέρονται στην Ενότητα 5.3. Εξάλλου, το έγγραφο περιγραφής του σχεδίου του λογισμικού (Σχήμα 5.6) δεν είναι παρά μια δομημένη διάταξη των αποτελεσμάτων των επιμέρους εργασιών της σχεδίασης. Η Ενότητα I αποτελεί εισαγωγική περιγραφή του εγγράφου και συμπληρώνεται ανεξάρτητα. Χρήσιμο είναι, μετά την ολοκλήρωση της συγγραφής του εγγράφου, να επανερχόμαστε στην Ενότητα I, προκειμένου να εξετάσουμε αν απαιτείται συμπλήρωση ορισμών ή αναφορών, για παράδειγμα. Όσοι δεν δώσατε την απάντηση αυτή, θα πρέπει να ανατρέξετε και πάλι στην Ενότητα 5.3.

### Άσκηση 3/Κεφάλαιο 5

Η σωστή απάντηση είναι η Β. Μια εφαρμογή λογισμικού, διατεταγμένη σύμφωνα με το σχήμα πελάτη – εξυπηρετητή, συνήθως λειτουργεί σε περιβάλλον δικτύου στο οποίο πολλά συστήματα τελικών χρηστών (πελάτες) μοιράζονται τις υπηρεσίες του εξυπηρετητή. Τα προβλήματα που προκύπτουν από τις επιδόσεις του δικτύου (επιλογή Α) παραμένουν σε όλες τις διατάξεις που χρησιμοποιούν δίκτυο. Μάλιστα, μπορεί να ισχυριστεί κανείς ότι το ίδιο δίκτυο ανταποκρίνεται καλύτερα σε κάποιες περιπτώσεις στη διάταξη πελάτη – εξυπηρετητή, διότι μέσω του δικτύου μεταφέρονται μη μορφοποιημένα δεδομένα, πράγμα που δεν συμβαίνει στη διάταξη του web client (στη γενική περίπτωση τα μορφοποιημένα δεδομένα καταλαμβάνουν περισσότερο όγκο).

Όταν συμβαίνουν μεταβολές στο λογισμικό κατά τον κύκλο ζωής του, η εφαρμογή λογισμικού που τρέχει στα συστήματα πελάτη αλλάζει. Αυτό δημιουργεί την απαίτηση συντήρησης των συστημάτων αυτών για ενημέρωσή τους με την τελευταία έκδοση. Η συντήρηση αυτή απαιτεί τόσο περισσότερο κόπο και χρόνο, όσο μεγαλύτερος είναι ο αριθμός των πελατών και η γεωγραφική τους κατανομή. Αυτός είναι και ο λόγος για τον οποίο το μειονέκτημα αυτό είναι και το σημαντικότερο. Στην περίπτωση του σχήματος στο οποίο χρησιμοποιείται web client, η ανάγκη αυτή δεν υπάρχει. Στο σύστημα

του πελάτη δεν λειτουργεί κανένα τμήμα της εφαρμογής, παρά μόνο ένας web browser. Όλη η συντήρηση γίνεται στα συστήματα των εξυπηρετητών και όχι στα πολυάριθμα συστήματα των πελατών, οπότε κοστίζει πολύ λιγότερο. Άρα, ορθώς διαλέξατε την επιλογή Β.

Όσοι κάνατε την επιλογή Γ μπορείτε να αναλογιστείτε το εξής: η πραγματοποίηση ενός υπολογισμού στοιχίζει σε υπολογιστικούς πόρους, στη γενική περίπτωση, περισσότερο απ' ό,τι η λειτουργία ενός web browser που απαιτείται στην πολυμερή διάταξη. Μπορεί, βέβαια, κανείς να δώσει παραδείγματα υπολογισμών απλούστερων από πλευράς υπολογιστικών απαιτήσεων απ' ό,τι η εμφάνιση, λόγου χάρη, κάποιων διαγραμμάτων. Στη γενική περίπτωση όμως, και για την ίδια εμφάνιση αποτελέσματος, η αρχιτεκτονική πελάτη – εξυπηρετητή επιβαρύνεται και με τον υπολογισμό του αποτελέσματος.

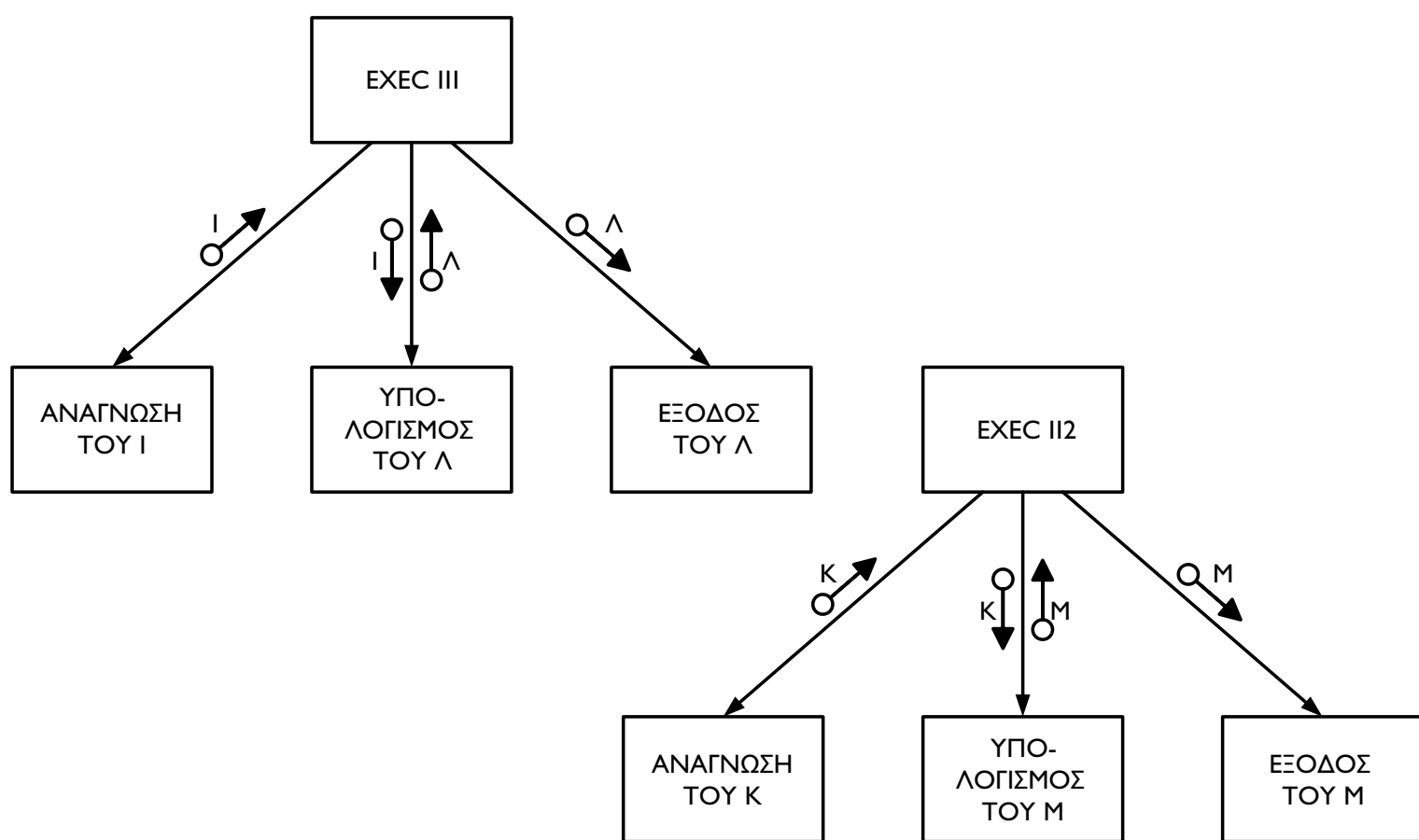
#### **Άσκηση 4/Κεφάλαιο 5**

---

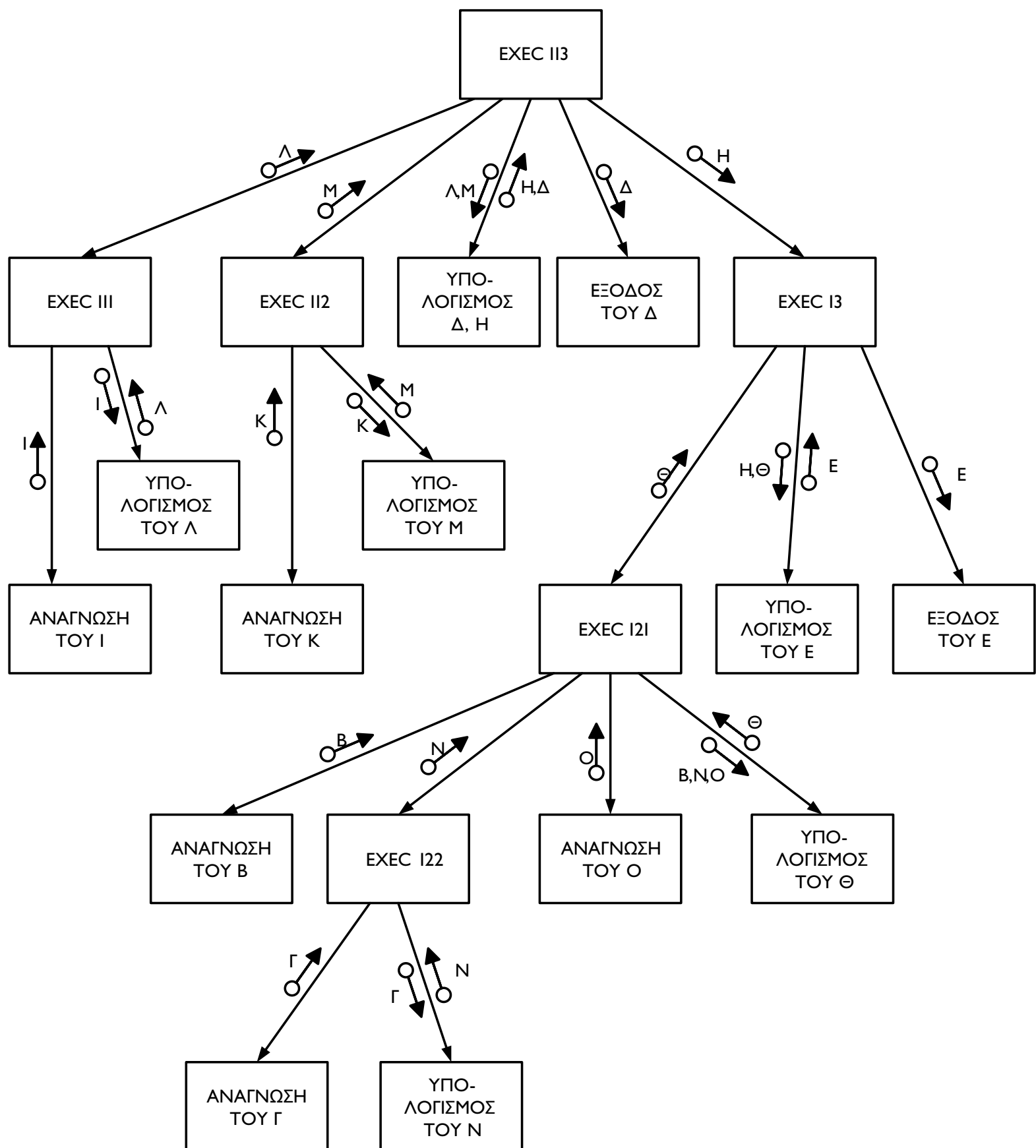
Τα ζητούμενα διαγράμματα φαίνονται ακολούθως. Στο Σχήμα 5.27 φαίνεται η παραγοντοποίηση για τους μετασχηματισμούς I.I.1 και I.I.2, ενώ στο Σχήμα 5.28 φαίνεται το πλήρες διάγραμμα δομής που προκύπτει μετά τη συνένωση.



**Σχήμα 5.27** Άσκηση αυτοαξιολόγησης 4.



**Σχήμα 5.28** Άσκηση αυτοαξιολόγησης 4.



Η παραγωγή των διαγραμμάτων είναι πολύ απλή και προκύπτει με άμεση εφαρμογή όσων αναφέρονται στην Ενότητα 5.5.2. Κάθε μετασχηματισμός έχει μόνο μία είσοδο και μία έξοδο, οπότε δεν έχουμε παρά απλή εφαρμογή του βήματος 3 της Ενότητας 5.5.2. Άξια προσοχής είναι η διαδικασία της συνένωσης: η μονάδα «ανάγνωση του  $\Lambda$ » που φαίνεται στο Σχήμα 5.17 δεν είναι παρά η ExecIII, διότι προϊόν αυτής είναι το  $\Lambda$ . Οπότε, κατά τη συγχώνευση, η ExecIII αντικαθιστά την «ανάγνωση του  $\Lambda$ », σύμφωνα με τα αναφερόμενα στο βήμα 4 της Ενότητας 5.5.2. Ακριβώς το ίδιο συμβαίνει κατά την συγχώνευση της ExecII2 (Σχήμα 5.28).

Δεν θα πρέπει να απογοητεύεστε αν τα «κουτάκια» σας δημιουργούν σύγχυση. Προσπαθήστε να αντιμετωπίσετε το θέμα ως την παράσταση λογικών πραγμάτων με σχήματα. Θα διαπιστώσετε ότι ακόμη και αν στην αρχή φαίνεται κουραστικό, τελικά είναι πολύ χρήσιμο. Μελετώντας ξανά το Παράδειγμα I θα βοηθηθείτε αρκετά.

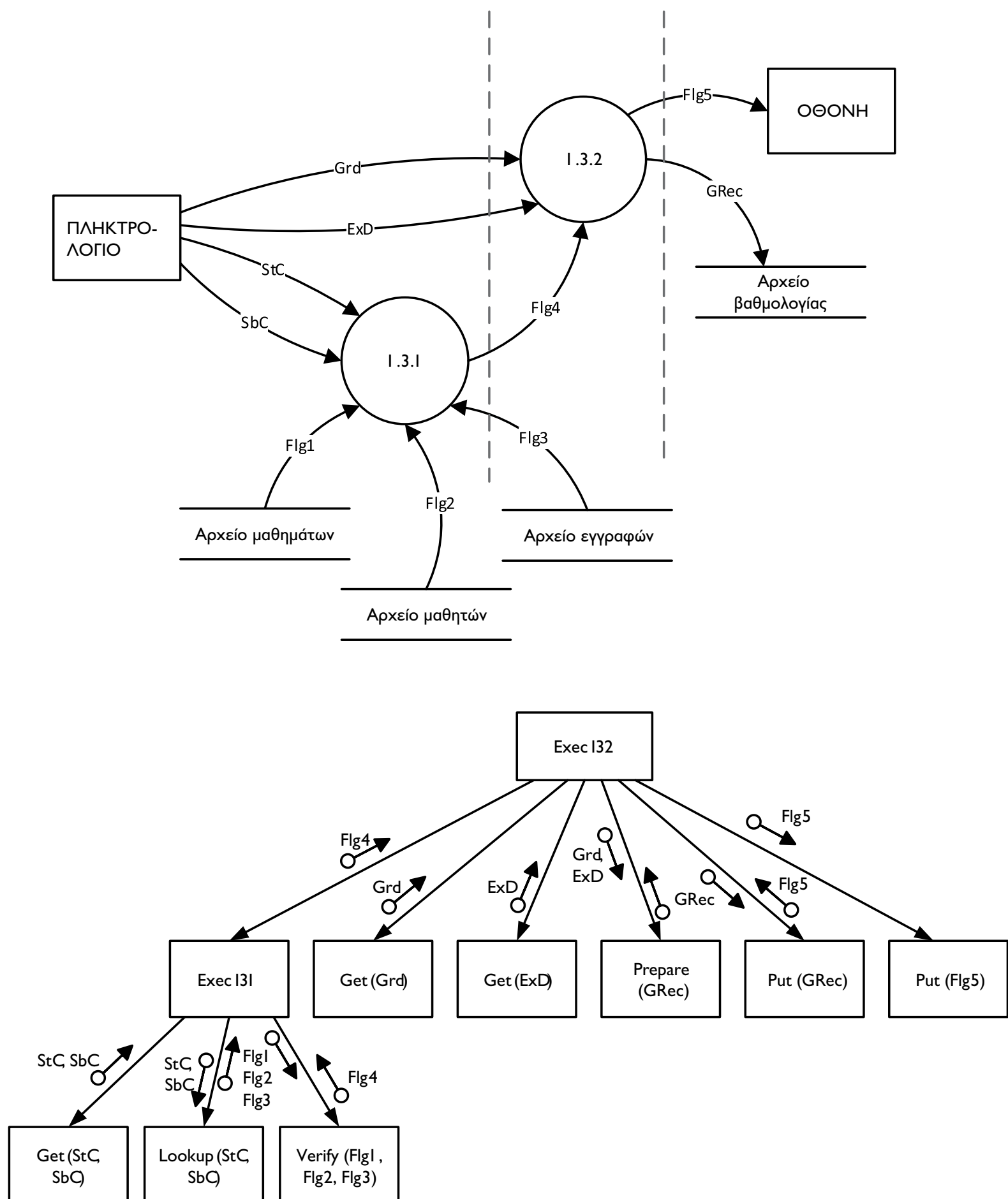
## Άσκηση 5/Κεφάλαιο 5

Θα πρέπει να εφαρμόσετε τα βήματα της Ενότητας 5.5.2 για τους εν λόγω μετασχηματισμούς. Τα πράγματα εδώ είναι κάπως πιο δύσκολα, μια και κάθε μετασχηματισμός έχει περισσότερες από μία εισόδους και εξόδους, οπότε ο αναγνώστης πρέπει να προσέξει την εφαρμογή των βημάτων της 5.5.2 συμβουλευόμενος το Παράδειγμα I και τη μελέτη περίπτωσης στην Ενότητα 5.5. Η συνένωση απαιτείται για τη μονάδα που αντιστοιχεί στο μετασχηματισμό I.3.1 και γίνεται με τον ίδιο ακριβώς τρόπο. Δοκιμάστε να δώσετε μόνοι σας τη λύση έχοντας κατανοήσει πλήρως τα παραδείγματα και τη μελέτη περίπτωσης της Ενότητας 5.5. Κατόπιν, συγκρίνετε τη λύση που δώσατε με αυτή που ακολουθεί. Αν τα αποτελέσματά σας συμπίπτουν με τα δικά μας, μπράβο! Αν αποκλίνετε, προσπαθήστε να επιβεβαιώσετε την εφαρμογή των βημάτων και ξαναπροσπαθήστε. Δεν είναι απλή περίπτωση και δεν πρέπει να σας προβληματίζει αν χρειαστεί να επαναλάβετε περισσότερες από μία φορές την προσπάθεια.

Μια απλοποιημένη εκδοχή του διαγράμματος ροής δεδομένων που περιέχει τους μετασχηματισμούς I.3.1 και I.3.2 του Σχήματος 4.12 μαζί με το

διάγραμμα δομής προγράμματος που αντιστοιχεί σε αυτό φαίνεται στο Σχήμα 5.29. Ο αναγνώστης παρακαλείται να συγχωρήσει τη χρήση συντμήσεων στην αγγλική, είναι όμως εύκολα αντιληπτό ότι αυτή επιβάλλεται για πρακτικούς λόγους σχεδίασης των διαγραμμάτων.

**Σχήμα 5.29** Άσκηση αυτοαξιολόγησης 5.



## ΒΙΒΛΙΟΓΡΑΦΙΑ

Boehm, B. W. (1975) In Horowitz, E. et al., *Practical Strategies for Developing Large Software Systems*, Reading, Massachusetts: Addison-Wesley.

Boehm, B. W., *Verifying and Validating Software Requirements and Design Specifications*, IEEE Software, January 1984, pp.75-88, 1984.

*IEEE Recommended Practice for Software Design Descriptions*, ANSI/IEEE, Std 1016-1987.

Peters, L. J. (1981) *Software Design*, New York: Yourdon Press. ISBN 0-91-707219-7.

Pressman, R. S., *Software Engineering-A Practitioners Approach*, McGraw-Hill.

Sommerville, I. *Software Engineering*, London: Addison-Wesley.

Yourdon, E. and Constantine, L. L. (1979) *Structured Design*, Englewood Cliffs, New Jersey: Prentice-Hall. ISBN 0-13-354471-9.

## ΠΑΡΑΓΩΓΗ ΠΗΓΑΙΟΥ ΚΩΔΙΚΑ

Σκοπός του κεφαλαίου είναι η παρουσίαση των εργασιών και των προβλημάτων κατά τη φάση συγγραφής του πηγαίου κώδικα του λογισμικού. Η παρουσίαση αυτή διατηρείται σε ένα αφαιρετικό, γενικό επίπεδο χωρίς να γίνεται αναφορά σε συγκεκριμένη γλώσσα προγραμματισμού.

Μετά τη μελέτη του κεφαλαίου αυτού, ο αναγνώστης θα είναι σε θέση:

- να διακρίνει τα εργαλεία συγγραφής πηγαίου κώδικα σε πέντε κατηγορίες,
- να διακρίνει τα επιθυμητά χαρακτηριστικά του πηγαίου κώδικα, ώστε να τα επιδιώκει κατά τη συγγραφή του,
- να αναγνωρίζει γενικά χαρακτηριστικά και είδη σύγχρονων γλωσσών προγραμματισμού,
- να αναγνωρίζει βασικές αιτίες σφαλμάτων στον πηγαίο κώδικα και να χρησιμοποιεί ένα σύνολο συμβουλών για παραγωγή πηγαίου κώδικα με λίγα σφάλματα,
- να αναφέρει τεχνικές αποφυγής σφαλμάτων και ανοχής σε σφάλματα.

### Έννοιες-κλειδιά

---

- *Επάρκεια πηγαίου κώδικα*
- *Αναγνωσιμότητα πηγαίου κώδικα*
- *Μεταφερσιμότητα πηγαίου κώδικα*
- *Δομημένος προγραμματισμός*
- *Αποφυγή σφαλμάτων*
- *Ανοχή σε σφάλματα*

Η παραγωγή του πηγαίου κώδικα είναι η φάση κατά την οποία το σχέδιο του λογισμικού μετατρέπεται σε πρόγραμμα. Πρόκειται για μια ιδιαίτερα κρίσιμη εργασία, διότι το σχέδιο σπάνια περιέχει όλες τις λεπτομέρειες και δεν λαμβάνει –συνήθως– υπόψη του τα ιδιαίτερα χαρακτηριστικά της γλώσσας προγραμματισμού που χρησιμοποιείται. Η Τεχνολογία Λογισμικού παρέχει ορισμένες κατευθυντήριες αρχές, ώστε ο πηγαίος κώδικας να έχει τα επιθυμητά χαρακτηριστικά της επάρκειας, των επιδόσεων, της αναγνωσιμότητας, της τεκμηρίωσης και της μεταφερσιμότητας, καθώς και κατευθύνσεις που βοηθούν στην αποφυγή σφαλμάτων. Η επαναχρησιμοποίηση μονάδων προγράμματος είναι μια κατεύθυνση στην οποία θα στραφεί η κοινότητα των κατασκευαστών λογισμικού στο μέλλον.

## Εισαγωγικές παρατηρήσεις

---

Η εργασία της κωδικοποίησης έπεται της σχεδίασης στην ανάπτυξη λογισμικού. Ως βάση για την εργασία τους, οι προγραμματιστές χρησιμοποιούν τα αποτελέσματα της σχεδίασης, τα οποία (πρέπει να) περιγράφουν σε ικανοποιητικό βαθμό λεπτομέρειας τον πηγαίο κώδικα που θα παραχθεί. Ένα μεγάλο μέρος της κωδικοποίησης θα μπορούσε να γίνεται στην πράξη με την άμεση μετατροπή του ψευδοκώδικα σε πηγαίο κώδικα, γραμμένο σε μια κατάλληλη γλώσσα προγραμματισμού. Πολύ συχνά, όμως, είναι απαραίτητη η προσθήκη κώδικα που δεν έχει προβλεφθεί κατά τη σχεδίαση για προγραμματιστικούς συνήθως λόγους, καθώς και η χρήση προγραμματιστικών τεχνικών των οποίων η περιγραφή ξεφεύγει από το σκοπό της σχεδίασης.

Επιπλέον, η κατασκευή του σχεδίου του λογισμικού δεν λαμβάνει πάντα υπόψη τα χαρακτηριστικά της γλώσσας προγραμματισμού που χρησιμοποιείται. Αυτό μπορεί να έχει ως επακόλουθο την ανάγκη ιδιαιτέρων ερμηνειών ή/και μεταφορών του ψευδοκώδικα σε πηγαίο κώδικα. Σε τελική ανάλυση, η ικανοποίηση των απαιτήσεων από το λογισμικό βρίσκεται στα χέρια αυτών που θα συγγράψουν τον πηγαίο κώδικα. Το γεγονός αυτό δεν πρέπει να διαφεύγει από κανένα μηχανικό λογισμικού: ο προγραμματισμός είναι ένας κρίκος στην αλυσίδα ανάπτυξης λογισμικού ο οποίος, αν και βρίσκεται πολύ κοντά στην άκρη της, μπορεί κάλλιστα να τη σπάσει.



## ΕΝΟΤΗΤΑ 6.1. ΑΠΟ ΤΗ ΣΧΕΔΙΑΣΗ ΣΤΗΝ ΚΩΔΙΚΟΠΟΙΗΣΗ

Η κατασκευή του πηγαίου κώδικα (κωδικοποίηση) είναι το τελευταίο μέρος της ανάπτυξης του λογισμικού το οποίο γίνεται, τουλάχιστον στο μεγαλύτερο μέρος του, από τον άνθρωπο. Με τη μετάβαση από τη σχεδίαση στην κωδικοποίηση ασχολείται η ενότητα αυτή.

### 6.1.1. Λογισμικό χωρίς σφάλματα

Η μετάβαση από τη λεπτομερή σχεδίαση στον πηγαίο κώδικα πρέπει να γίνεται με τρόπο που να παράγεται λογισμικό χωρίς σφάλματα. Η τελευταία έννοια γίνεται συχνά σημείο αναφοράς από πολλούς εμπλεκόμενους στην ανάπτυξη του λογισμικού και είναι σκόπιμο να οριστεί ο τρόπος με τον οποίο θα τη χρησιμοποιούμε στη συνέχεια.

#### Λογισμικό χωρίς σφάλματα:

Είναι το λογισμικό που πληροί τις προδιαγραφές βάσει των οποίων κατασκευάστηκε, χωρίς να παράγει σφάλματα κατά την εκτέλεσή του (runtime), δηλαδή το λογισμικό που κάνει ακριβώς αυτό για το οποίο προορίζεται και το κάνει σωστά.

Όπως έχει ήδη αναφερθεί, οι προδιαγραφές αυτές μπορεί να είναι εσφαλμένες, ανεπίκαιρες και, γενικά, να μην αντιστοιχούν στις πραγματικές ανάγκες του χρήστη. Αυτό είναι ένα πρόβλημα που γεννιέται και οφείλει να αντιμετωπίζεται σε προηγούμενες φάσεις της ανάπτυξης και όχι στην κωδικοποίηση. Το λογισμικό χωρίς σφάλματα δεν συμπεριφέρεται απαραίτητα κατά τον τρόπο που αναμένουν οι χρήστες, ακριβώς επειδή ενδέχεται να μην είναι ακριβείς οι προδιαγραφές.

Οι λόγοι για τους οποίους μπορεί να συμβαίνει αυτό έχουν αναφερθεί σε αρκετά σημεία του βιβλίου που κρατάτε. Οι προδιαγραφές μπορεί να μεταβληθούν κατά την ανάπτυξη του λογισμικού και τη στιγμή που φτάνουμε στη συγγραφή του πηγαίου κώδικα να μην ισχύουν κάποιες από αυτές. Όπως χαρακτηριστικά αναφέρεται και στην Ενότητα 1.2, «η ανάπτυξη του

λογισμικού ομοιάζει με σκόπευση κινουμένου στόχου από κινούμενο έδαφος και με όπλο που συνεχώς αλλάζει τη συμπεριφορά του».

Είναι ο κόσμος που μεταβάλλεται και η διαλεκτική αλληλεπίδραση όλων των συστημάτων του που δημιουργεί συνεχώς, ακόμα και πριν ολοκληρωθεί η ανάπτυξη μιας εφαρμογής, αιτίες τροποποιήσεων στο λογισμικό (Ενότητες I.4 και I.5, Σχήμα I.2). Σε αυτό προστίθενται οι εγγενείς και πραγματικές δυσκολίες στην περιγραφή των απαιτήσεων από το λογισμικό (Ενότητα 4.5) και, τελικά, μπορεί να καθίστανται ανακριβείς, εσφαλμένες ή, τέλος πάντων, ανεπίκαιρες κάποιες από τις προδιαγραφές του λογισμικού κατά την έναρξη της συγγραφής του πηγαίου κώδικα. Εδώ βρίσκεται και η πρόκληση που πρέπει να αντιμετωπίσει ο μηχανικός λογισμικού. Η Τεχνολογία Λογισμικού δεν λύνει από μόνη της όλα τα προβλήματα ούτε μπορεί να υποσχεθεί κάτι τέτοιο. Μπορεί όμως να αποτελέσει ένα χρήσιμο επιστημονικό εργαλείο για την αντιμετώπιση του προβλήματος. Σήμερα είναι το μόνο τέτοιο εργαλείο που διαθέτουμε.

Η ανάπτυξη λογισμικού χωρίς σφάλματα είναι λοιπόν ιδιαίτερα δύσκολη και στην πράξη μάλλον ανέφικτη. Η εμφάνιση των σύγχρονων γλωσσών και περιβαλλόντων προγραμματισμού, που διευκολύνουν την ανάπτυξη σύνθετου και καλά δομημένου κώδικα, καθώς και η βελτίωση των χρησιμοποιούμενων προγραμματιστικών τεχνικών οδηγούν γενικά σε βελτίωση της ποιότητας του λογισμικού.

### **6.1.2. Εργαλεία κωδικοποίησης**

Ένα μεγάλο πλήθος εργαλείων έχει κατασκευαστεί για τη διευκόλυνση των προγραμματιστών στη φάση της κωδικοποίησης. Τα πρωταρχικά εργαλεία στη φάση αυτή είναι φυσικά οι γλώσσες προγραμματισμού και γύρω από αυτές περιστρέφεται ένας μεγάλος αριθμός επιμέρους εργαλείων. Στο Σχήμα 6.1 γίνεται μια κατηγοριοποίηση των κυριότερων εργαλείων που είναι στη διάθεση των προγραμματιστών κατά τη φάση της κωδικοποίησης, οι οποίες κατηγορίες επεξηγούνται στη συνέχεια.

**Σχήμα 6.1** Εργαλεία κωδικοποίησης.



**Συντάκτες προγραμμάτων** (program editors). Είναι εξειδικευμένα προγράμματα επεξεργασίας κειμένου που αποσκοπούν στη διευκόλυνση της πρωτογενούς εργασίας συγγραφής κώδικα. Αρκετοί από τους σύγχρονους συντάκτες προγραμμάτων υποστηρίζουν έλεγχο της σύνταξης σε μία ή περισσότερες γλώσσες προγραμματισμού και λειτουργίες όπως χρωματική σύνταξη (syntax highlighting, δηλαδή χρωματισμό των δεσμευμένων λέξεων, των σχολίων, των μεταβλητών κ.λπ. με διαφορετικά χρώματα), αυτόματη στοίχιση, αρίθμηση γραμμών κ.ά.

**Μεταφραστικά περιβάλλοντα γλωσσών προγραμματισμού** (programming language implementations) και παρεμφερή εργαλεία. Συνήθως περιέχουν ένα ή περισσότερα από τα παρακάτω είδη προγραμμάτων, καθώς και άλλα βοηθητικά προγράμματα. Αξίζει να σημειωθεί ότι σήμερα τέτοια εργαλεία υπάρχουν ως ανεξάρτητα προγράμματα μόνο σε παλαιά ή ειδικής χρήσης υπολογιστικά συστήματα. Τα σύγχρονα μεταφραστικά περιβάλλοντα προγραμματισμού περιέχουν διάφορα εργαλεία κάτω από ένα ενιαίο κέλυφος, όπως αναφέρεται παρακάτω:

- *Μεταγλωττιστές* (compilers). Μεταφράζουν πηγαίο κώδικα γραμμένο σε μια γλώσσα προγραμματισμού υψηλού επιπέδου σε εκτελέσιμο κώδικα μηχανής.
- *Διερμηνείς* (interpreters). Εκτελούν γραμμή προς γραμμή πηγαίο κώδικα γραμμένο σε μια γλώσσα προγραμματισμού υψηλού επιπέδου.
- *Συμβολομεταφραστές* (assemblers). Μεταφράζουν πηγαίο κώδικα γραμμένο σε μια συμβολική γλώσσα μηχανής (assembly language) σε εκτελέσιμο κώδικα μηχανής.

**Εντοπιστές σφαλμάτων** (debuggers). Βοηθούν στον εντοπισμό σφαλμάτων κατά την εκτέλεση των προγραμμάτων. Υποστηρίζουν λειτουργίες όπως η εκτέλεση βήμα προς βήμα, η εμφάνιση τιμών μεταβλητών, η εμφάνιση της στοίβας εκτέλεσης και ο καθορισμός σημείων και συνθηκών διακοπής στην εκτέλεση του κώδικα.

**Γεννήτορες προγραμμάτων** (program generators). Πρόκειται για προγράμματα που δέχονται ως είσοδο τις προδιαγραφές του κώδικα γραμμένες σε

κάποια γλώσσα υψηλού επιπέδου και παράγουν πηγαίο κώδικα γραμμένο σε κάποια γλώσσα προγραμματισμού, που να πληροί τις δοθείσες προδιαγραφές. Οι γεννήτορες προγραμμάτων δεν είναι γενικοί και συνήθως εξειδικεύονται σε στενές θεματικές κατηγορίες προβλημάτων όπως, για παράδειγμα, η κατασκευή προγραμμάτων που υλοποιούν τη διεπαφή ανθρώπου – υπολογιστή, η κατασκευή λεκτικών και συντακτικών αναλυτών, προγραμμάτων προσομοίωσης κ.ά.

**Συστήματα υποστήριξης λογισμικού** (software support systems) και **ολοκληρωμένα περιβάλλοντα προγραμματισμού** (integrated programming environments). Αποτελούν προσπάθειες συγκέντρωσης και ενοποίησης διαφόρων εργαλείων προγραμματισμού που συνεργάζονται μεταξύ τους. Τα ολοκληρωμένα αυτά εργαλεία συνήθως περιλαμβάνουν έναν συνδυασμό όλων των χρήσιμων για τον προγραμματιστή εργαλείων, κοινός παρονομαστής των οποίων είναι συνήθως μια γλώσσα προγραμματισμού. Τα περισσότερα σύγχρονα περιβάλλοντα υλοποίησης γλωσσών προγραμματισμού ανήκουν σε αυτή την κατηγορία.

### Δραστηριότητα I/Κεφάλαιο 6

Είναι άμεση η μετάβαση από τον ψευδοκώδικα στον πηγαίο κώδικα; Περιγράψτε σε 50-100 λέξεις τις σημαντικότερες αιτίες για την επιπλέον δουλειά που καλείται να κάνει ο προγραμματιστής.

### Άσκηση I/Κεφάλαιο 6

Αναφέρατε τις πέντε σημαντικότερες κατηγορίες εργαλείων που έχει στη διάθεσή του ο προγραμματιστής, προκειμένου να κάνει τη δουλειά του.

## ΕΝΟΤΗΤΑ 6.2. ΕΠΙΘΥΜΗΤΑ ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΟΥ ΠΗΓΑΙΟΥ ΚΩΔΙΚΑ

Στην ενότητα αυτή θα αναφερθούν τα χαρακτηριστικά του πηγαίου κώδικα που είναι επιθυμητά, προκειμένου να εξασφαλίζεται η ποιότητα του λογισμικού. Κατά τη κωδικοποίηση δεν αρκεί να κατασκευάζεται κάποιος οποιοσδήποτε πηγαίος κώδικας, αλλά είναι σημαντικό να τηρούνται τα απαραίτητα μέτρα, ώστε ο πηγαίος κώδικας να διαθέτει αυτά τα επιθυμητά χαρακτηριστικά.

### 6.2.1. Επάρκεια

Το βασικότερο επιθυμητό χαρακτηριστικό του πηγαίου κώδικα είναι η επάρκεια (efficiency). Ο κώδικας χαρακτηρίζεται επαρκής όταν το σύστημα λογισμικού λειτουργεί σωστά και χωρίς σφάλματα βάσει των απαιτήσεων που έχουν καθοριστεί, κάνει σωστή χρήση των διατιθέμενων πόρων και είναι διαθέσιμο όποτε απαιτείται. Η επάρκεια του πηγαίου κώδικα εξασφαλίζεται κυρίως κατά τη φάση ελέγχου του λογισμικού, με την οποία θα ασχοληθούμε στο Κεφάλαιο 7.

### 6.2.2. Επίδοση

Σε πολλά συστήματα λογισμικού καθοριστικό ρόλο κατέχει η επίδοση (performance). Στη μέτρηση της επίδοσης λαμβάνονται συνήθως υπόψη τα ακόλουθα χαρακτηριστικά:

- Η ταχύτητα εκτέλεσης συγκεκριμένων λειτουργιών του συστήματος λογισμικού.
- Οι απαιτήσεις του συστήματος σε πόρους όπως μνήμη και χωρητικότητα δίσκου.

Όπως είναι φυσικό, διαφορετικά κομμάτια πηγαίου κώδικα που ικανοποιούν τις ίδιες λειτουργικές προδιαγραφές είναι δυνατό να διαφέρουν πολύ στην επίδοση, ακόμα κι αν υλοποιούν τον ίδιο αλγόριθμο. Η επιλογή της κατάλληλης γλώσσας προγραμματισμού αλλά και η χρήση κατάλληλων



προγραμματιστικών τεχνικών οδηγούν σε βελτίωση της επίδοσης του τελικού συστήματος λογισμικού.

### 6.2.3. Αναγνωσιμότητα

Στη διαδικασία ανάπτυξης λογισμικού συμβαίνει πολύ συχνά να χρειαστεί ένας προγραμματιστής να διαβάσει και να κατανοήσει τον πηγαίο κώδικα που έχει γράψει ένας άλλος προγραμματιστής. Συμβαίνει επίσης αρκετά συχνά να δυσκολεύεται ένας προγραμματιστής να καταλάβει κώδικα που έχει γράψει ο ίδιος ύστερα από λίγο καιρό. Για τους δύο αυτούς λόγους, αναγκαίο χαρακτηριστικό του πηγαίου κώδικα είναι η αναγνωσιμότητα (readability), δηλαδή η ευκολία κατανόησης του πηγαίου κώδικα αποκλειστικά μέσω της ανάγνωσής του ως κειμένου. Ακολουθούν ορισμένες συμβουλές για τη συγγραφή πιο αναγνώσιμου κώδικα, ανεξαρτήτως γλώσσας προγραμματισμού.

- Απλότητα. Είναι καλό οι προγραμματιστές να αποφεύγουν την περιττή πολυπλοκότητα στη συγγραφή του πηγαίου κώδικα, όταν αυτό δεν είναι αναγκαίο για την εξασφάλιση των απαιτήσεων επίδοσης. Η αναζήτηση απλών και κομψών λύσεων στην κωδικοποίηση δεν είναι καθόλου ευκαταφρόνητο εγχείρημα. Επιπλέον, ο απλούστερος κώδικας, εκτός από περισσότερο ευανάγνωστος, είναι και λιγότερο επιρρεπής σε σφάλματα.
- Πλεονασμός χάρη σαφήνειας. Η χρήση κατάλληλων σημείων στίξης οδηγεί σε πιο ευανάγνωστο κώδικα, όπως ακριβώς και στη φυσική γλώσσα. Τέτοια σημεία στίξης είναι κυρίως οι παρενθέσεις και οι λέξεις-κλειδιά για την ομαδοποίηση εντολών (begin και end, { και } κ.ά.) που, αν και ορισμένες φορές είναι περιττές σύμφωνα με τους συντακτικούς κανόνες της γλώσσας προγραμματισμού, είναι καλό να χρησιμοποιούνται για τη διευκόλυνση του αναγνώστη.
- Επιλογή κατάλληλων ονομάτων. Εξαιρετικής σημασίας για τη συγγραφή αναγνώσιμου κώδικα είναι η σωστή επιλογή ονομάτων για τα αναγνωριστικά του προγράμματος. Το ακόλουθο παράδειγμα, στο οποίο τα δύο κομμάτια κώδικα σε γλώσσα C είναι ισοδύναμα και διαφέρουν μόνο στα ονόματα των μεταβλητών και στη χρήση σταθερών, είναι ιδιαίτερα πειστικό.

```

if (x > 50000)    const double NOT_TAXABLE = 50000;
    y = (x - 50000) * 0.2;    const double TAX_RATE = 0.2;
Else
    y = 0;
if (income > NOT_TAXABLE)
    tax = (income - NOT_TAXABLE) * TAX_RATE;
else
    tax = 0;

```

- Χρήση σχολίων. Τα σχόλια αποτελούν αναπόσπαστο μέρος του πηγαίου κώδικα και, ως εκ τούτου, πρέπει να γράφονται συγχρόνως με τον κώδικα και να ενημερώνονται ανελλιπώς σε κάθε μετέπειτα αλλαγή του. Τις περισσότερες φορές η έλλειψη σχολίων δυσχεραίνει ή καθιστά εντελώς αδύνατη την κατανόηση ενός προγράμματος. Για το λόγο αυτό, περιγραφικά σχόλια πρέπει να γράφονται τουλάχιστον στην αρχή των μονάδων του πηγαίου κώδικα αλλά και όπου απαιτείται στο εσωτερικό αυτών.
- Στοίχιση. Η αναγνωσιμότητα του πηγαίου κώδικα βελτιώνεται κατά πολύ αν χρησιμοποιούνται κανόνες στοίχισης του κώδικα και περιορίζεται ο αριθμός εντολών ανά γραμμή προγράμματος. Η μορφή της στοίχισης είναι υποκειμενική και πρέπει να αποφασίζεται από κοινού από την ομάδα κωδικοποίησης. Στη συνέχεια όμως, οι κανόνες στοίχισης πρέπει να χρησιμοποιούνται πιστά από όλα τα μέλη της ομάδας.

#### 6.2.4. Τεκμηρίωση

Οι τεχνικές για τη βελτίωση της αναγνωσιμότητας που αναφέρθηκαν παραπάνω δεν είναι πάντα αρκετές ως βοηθήματα για την κατανόηση του πηγαίου κώδικα. Είναι συχνά αναγκαίο να τεκμηριώνεται ο πηγαίος κώδικας με τη βοήθεια συνοδευτικών εγγράφων γραμμένων σε κατάλληλη γλώσσα. Η τεκμηρίωση (documentation) του πηγαίου κώδικα ποικίλει σε έκταση, μορφή και αυστηρότητα της γλώσσας που χρησιμοποιείται. Στην πράξη,



χρησιμοποιείται συνήθως φυσική γλώσσα που αποσκοπεί στην επεξήγηση των πιο πολύπλοκων σημείων του πηγαίου κώδικα, όταν αυτό δεν είναι εύκολο να γίνει με σχόλια μέσα σε αυτόν.

### 6.2.5. Μεταφερσιμότητα

Με τον όρο «μεταφερσιμότητα» (portability) εννοούμε την ιδιότητα του πηγαίου κώδικα να μπορεί να εκτελεστεί χωρίς αλλαγές σε διαφορετικά περιβάλλοντα (λειτουργικά συστήματα, τύπος υπολογιστή) στα οποία διατίθενται εκδόσεις της γλώσσας προγραμματισμού που χρησιμοποιείται. Από την πλευρά του μηχανικού λογισμικού είναι συνήθως πολύ σημαντικό να μπορεί να μεταφερθεί ο πηγαίος κώδικας ως έχει από μία υλοποίηση της γλώσσας προγραμματισμού στην οποία είναι γραμμένος σε μία άλλη. Για να επιτευχθεί αυτό, τις περισσότερες φορές είναι αρκετό:

- να μη χρησιμοποιούνται συστατικά της γλώσσας που υποστηρίζονται μόνο από συγκεκριμένες υλοποιήσεις και
- να μη γίνονται αυθαίρετες παραδοχές για τη συμπεριφορά των υλοποιήσεων, δηλαδή παραδοχές που δεν περιγράφονται ρητά στο πρότυπο (standard) της γλώσσας.

### 6.2.6. Δυνατότητα επαναχρησιμοποίησης

Η δυνατότητα επαναχρησιμοποίησης (reusability) είναι ιδιαίτερα σημαντικό χαρακτηριστικό του πηγαίου κώδικα, καθώς ένα αρκετά μεγάλο τμήμα του πηγαίου κώδικα ενός συστήματος λογισμικού μπορεί συνήθως να χρησιμοποιηθεί στην κωδικοποίηση άλλων συστημάτων λογισμικού. Προϋπόθεση για τη συγγραφή επαναχρησιμοποιήσιμου κώδικα είναι η κατάλληλη δόμησή του. Το θέμα της επαναχρησιμοποίησης πηγαίου κώδικα συζητείται στην Ενότητα 6.5.

## Άσκηση 2/Κεφάλαιο 6

Αναφέρατε εν συντομία τα έξι επιθυμητά χαρακτηριστικά του πηγαίου κώδικα δίνοντας έναν τίτλο και μια περιγραφή I-2 γραμμών για το καθένα.

## Δραστηριότητα 2/Κεφάλαιο 6

Αναφέρατε δύο τουλάχιστον λόγους για τους οποίους η αναγνωσιμότητα είναι ιδιαίτερα επιθυμητό χαρακτηριστικό του πηγαίου κώδικα.

## ΕΝΟΤΗΤΑ 6.3. ΓΛΩΣΣΕΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ

### 6.3.1. Εισαγωγή

Η γλώσσα προγραμματισμού είναι το βασικό εργαλείο του προγραμματιστή, ένα σύστημα συμβολισμών με το οποίο μπορεί να περιγράφει εργασίες που πρόκειται να εκτελεστούν από έναν υπολογιστή. Η ποικιλία των εργασιών που είναι επιθυμητό να εκτελούνται από τους υπολογιστές δικαιολογεί την ύπαρξη πληθώρας γλωσσών προγραμματισμού που διαφέρουν στην πολυπλοκότητα, στη μορφή, στη θεματική περιοχή όπου απευθύνονται αλλά και στη φιλοσοφία προγραμματισμού που υποστηρίζουν.

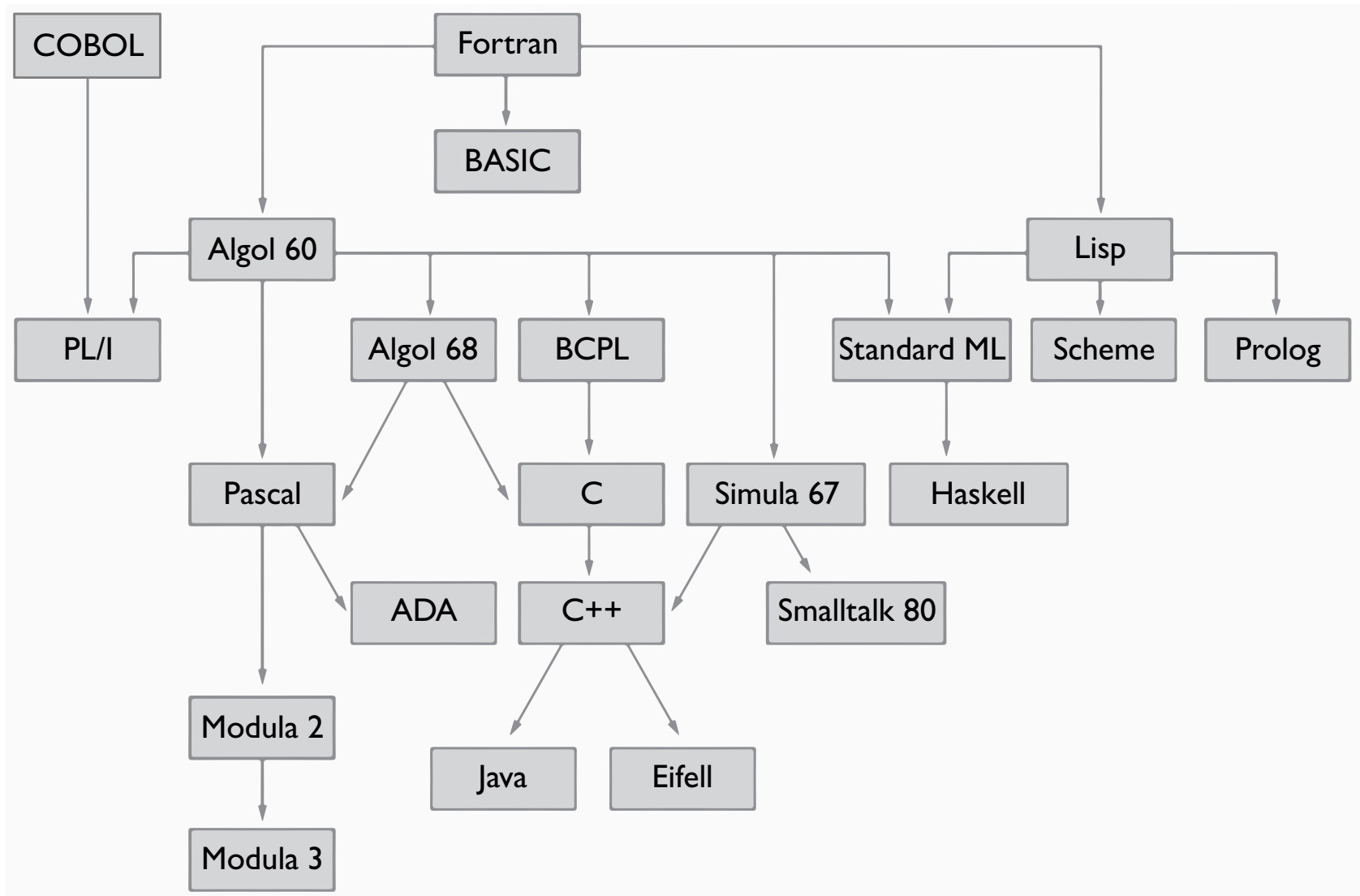
Οι γλώσσες προγραμματισμού «χαμηλού επιπέδου» ονομάζονται έτσι γιατί είναι κοντά στη μονάδα επεξεργασίας του υπολογιστή και μοιάζουν με τη γλώσσα μηχανής αυτού. Το μικροσκοπικό ύφος προγραμματισμού που επιβάλλουν, όμως, διαφέρει πολύ από τον τρόπο σκέψης των ανθρώπων και από τον τρόπο με τον οποίο αυτοί περιγράφουν τη λύση προβλημάτων. Για το λόγο αυτό αναπτύχθηκαν οι γλώσσες προγραμματισμού «υψηλού

επιπέδου», που χρησιμοποιούν συμβολισμούς περισσότερο αφαιρετικούς και πλησιέστερους στην ανθρώπινη αντίληψη.

Με τη χρήση γλωσσών χαμηλού επιπέδου είναι πρακτικά αδύνατο να εξασφαλιστεί η απαιτούμενη ποιότητα του λογισμικού, διότι τα τεχνικά προβλήματα που θα καλούταν να αντιμετωπίσει ο κατασκευαστής θα ήταν εντελώς ξένα με το πεδίο της εφαρμογής. Ως εκ τούτου, οι γλώσσες χαμηλού επιπέδου χρησιμοποιούνται μόνο για την υλοποίηση τμημάτων ενός συστήματος λογισμικού που είναι εξαιρετικά απαιτητικό από πλευράς επιδόσεων ή που αφορά συγκεκριμένες συσκευές. Τέτοια τμήματα είναι συνήθως μέρη λειτουργικών συστημάτων ή λογισμικό ειδικών απαιτήσεων, όπως αυτό που σχετίζεται με συστήματα πραγματικού χρόνου (real time systems).

Στη φάση της κωδικοποίησης προτιμώνται γλώσσες προγραμματισμού υψηλού επιπέδου, όπως η Pascal, η C, η C++, παλιότερα η COBOL και η BASIC και πιο πρόσφατα η Java, οι οποίες προσεγγίζουν τη χρησιμοποιούμενη φιλοσοφία ανάλυσης και σχεδίασης και διαθέτουν χαρακτηριστικά αναγκαία για την εξασφάλιση της ποιότητας του παραγόμενου λογισμικού. Στο Σχήμα 6.2 δίνεται μια συνολική εικόνα της εξέλιξης των κυριότερων γλωσσών προγραμματισμού που χρησιμοποιήθηκαν ή χρησιμοποιούνται σήμερα στην ανάπτυξη λογισμικού, καθώς και ορισμένων νέων γλωσσών που αναμένεται να γνωρίσουν διάδοση στο άμεσο μέλλον.

**Σχήμα 6.2** Εξέλιξη των γλωσσών προγραμματισμού.



### 6.3.2. Δομημένος προγραμματισμός

Οι αρχές του δομημένου προγραμματισμού γεννήθηκαν μέσα από μια επιστημονική διαμάχη στο τέλος της δεκαετίας του '60, με θέμα τις δομές ελέγχου των γλωσσών προγραμματισμού. Το αποτέλεσμα της διαμάχης ήταν ο σταδιακός παραγκωνισμός της χρήσης της εντολής `goto` και η αντικατάστασή της από δομημένες εντολές `if` και `while`.

#### Δομημένος προγραμματισμός (structured programming):

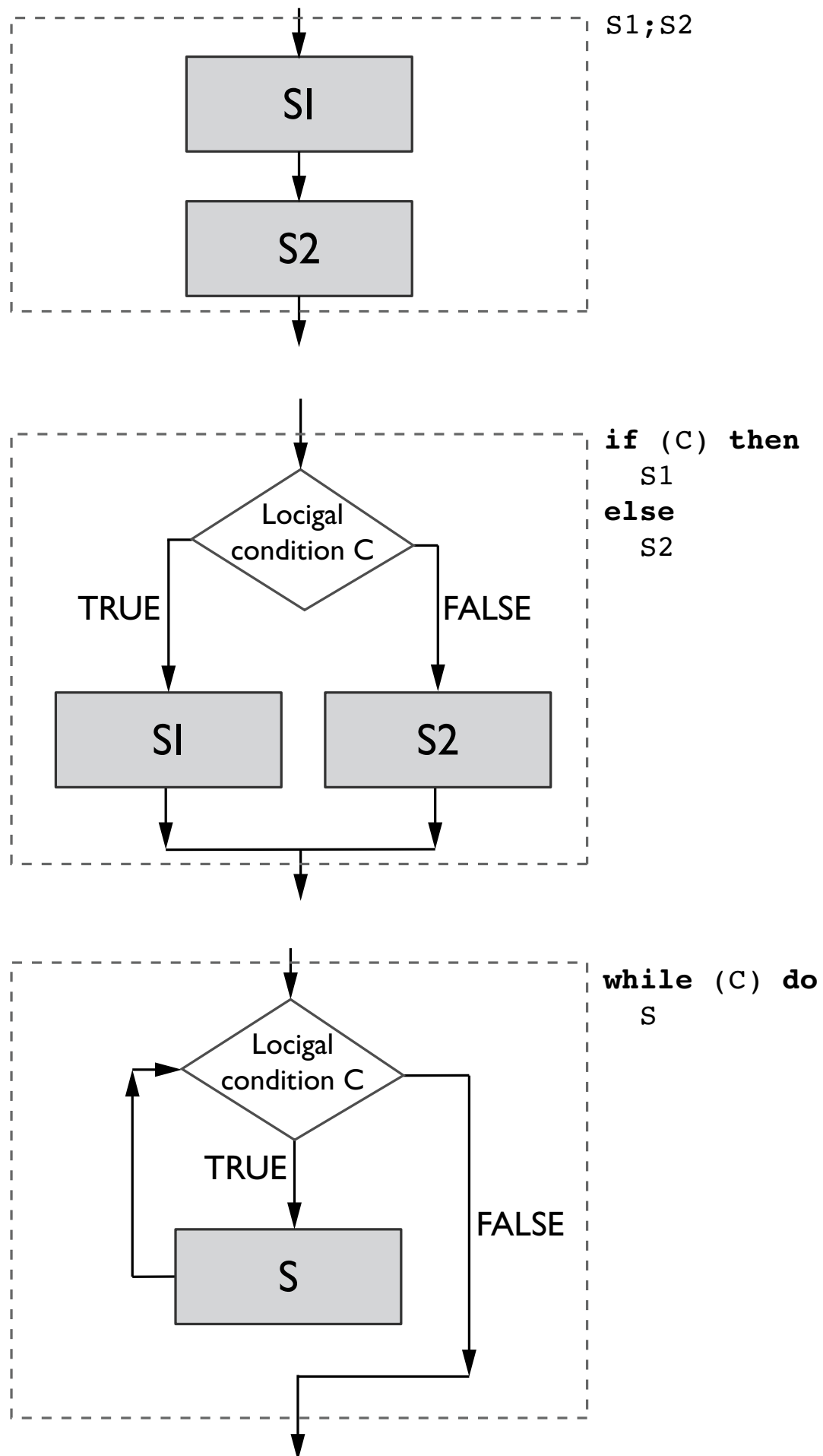
Η τεχνοτροπία προγραμματισμού κατά την οποία η συντακτική δομή του προγράμματος βοηθάει στην κατανόηση της ροής ελέγχου και, κατ' επέκταση, στην κατανόηση του τι κάνει το πρόγραμμα αυτό.

Αποδεικνύεται ότι κάθε αλγόριθμος είναι δυνατό να κωδικοποιηθεί χρησιμοποιώντας τρία μόνο στοιχεία ελέγχου:

- Ακολουθία (`s1; s2`)
- Επιλογή (`if c then s1 else s2`)
- Επανάληψη (`while c do s`)

Τα στοιχεία αυτά αποτελούν τη βάση του δομημένου προγραμματισμού. Όπως φαίνεται στο Σχήμα 6.3, χαρακτηρίζονται από την ύπαρξη ενός και μόνο σημείου εισόδου και ενός σημείου εξόδου, κάτι που επιτρέπει τον συνδυασμό τους χωρίς περιορισμούς σε μεγαλύτερες δομικές μονάδες.

**Σχήμα 6.3** Βασικά γλωσσικά στοιχεία του δομημένου προγραμματισμού.



Η εφαρμογή αυστηρής πειθαρχίας στη χρήση αυτών των βασικών στοιχείων και η αποφυγή της χρήσης γλωσσικών στοιχείων που διαχωρίζουν τη ροή ελέγχου από τη σύνταξη του προγράμματος ενισχύουν την αναγνωσιμότητα του πηγαίου κώδικα. Χωρίς να ξεφεύγουν από αυτό το πλαίσιο, πολλές μοντέρνες γλώσσες προγραμματισμού υποστηρίζουν και άλλα γλωσσικά στοιχεία που συνήθως μπορούν να υλοποιηθούν εύκολα από τα βασικά στοιχεία.

Στην πράξη όμως, ορισμένες φορές παρουσιάζεται η ανάγκη να παραβιαστεί η αποκλειστική χρήση των τριών βασικών στοιχείων του δομημένου προγραμματισμού. Χαρακτηριστικά παραδείγματα που εμφανίζονται συχνά είναι η ανάγκη άμεσης εξόδου από έναν βρόχο ή από μία συνάρτηση, καθώς και η άμεση μεταφορά της ροής ελέγχου ως συνέπεια της ανάγκης χειρισμού σφαλμάτων μέσα στον πηγαίο κώδικα. Οι σύγχρονες γλώσσες προγραμματισμού διαθέτουν, συνήθως, τέτοια γλωσσικά στοιχεία, μεριμνούν όμως ώστε η χρήση αυτών να ενισχύει την αναγνωσιμότητα του πηγαίου κώδικα αντί να τη δυσχεραίνει.

Η προγραμματιστική πειθαρχία που επιβάλλει ο δομημένος προγραμματισμός υποχρεώνει τον προγραμματιστή να σκέφτεται περισσότερο πριν αρχίσει να γράφει ένα κομμάτι κώδικα, και αυτό συντείνει στη μείωση της πιθανότητας σφαλμάτων. Με τη χρήση κατάλληλων γλωσσών που υποστηρίζουν διαδικασίες και συναρτήσεις ως δομικές μονάδες προγράμματος, ο δομημένος προγραμματισμός διευκολύνει την ανάπτυξη από πάνω προς τα κάτω (top-down) και υποστηρίζει τη φιλοσοφία της δομημένης ανάλυσης και σχεδίασης συστημάτων λογισμικού η οποία παρουσιάστηκε.

### Άσκηση 3/Κεφάλαιο 6

**Ποιες είναι οι τρεις δομές ελέγχου με χρήση των οποίων μπορεί να κωδικοποιηθεί κάθε αλγόριθμος στον δομημένο προγραμματισμό;**

### 6.3.3. Χαρακτηριστικά σύγχρονων γλωσσών προγραμματισμού

#### Συστήματα τύπων

Το σύστημα τύπων (type system) κατέχει εξέχουσα θέση στις σύγχρονες γλώσσες προγραμματισμού. Συγκεκριμένα, το σύστημα τύπων κάθε γλώσσας καθορίζει:

- τους τύπους δεδομένων που υποστηρίζει η γλώσσα,
- τις πράξεις που επιτρέπονται με τα δεδομένα αυτών των τύπων και
- το σύνολο των κανόνων που σε κάθε έκφραση της γλώσσας αντιστοιχούν σε έναν μοναδικό τύπο.

Οι πρώτες γλώσσες που αναπτύχθηκαν είχαν πολύ απλά συστήματα τύπων, ενώ τα συστήματα τύπων των γλωσσών μηχανής είναι σχεδόν πάντα τετριμμένα. Οι σύγχρονες γλώσσες προγραμματισμού υψηλού επιπέδου διαθέτουν συνήθως πολύπλοκα συστήματα τύπων που βοηθούν σημαντικά τον προγραμματιστή στη συγγραφή καλά δομημένου και ευανάγνωστου πηγαίου κώδικα.

#### Ισχυρό σύστημα τύπων:

Ένα σύστημα τύπων ονομάζεται «ισχυρό» (strong) όταν επιτρέπει μόνο εκφράσεις των οποίων η αποτίμηση δεν προκαλεί σφάλματα εκτέλεσης. Τα ισχυρά συστήματα τύπων εξασφαλίζουν ότι η αποτίμηση των εκφράσεων είναι ασφαλής.

Οι κανόνες ενός συστήματος τύπων μπορούν να χρησιμοποιηθούν για τον έλεγχο της σωστής εφαρμογής των τελεστών και, κατ' επέκταση, της σωστής σύνταξης των εκφράσεων της γλώσσας. Ο έλεγχος αυτός γίνεται ως επί το πλείστον κατά τη μεταγλώττιση του προγράμματος και σε σπανιότερες περιπτώσεις κατά την εκτέλεσή του. Η ύπαρξη ενός πλούσιου και ισχυρού συστήματος τύπων σε μια γλώσσα προγραμματισμού ενισχύει την αναγνωσιμότητα του πηγαίου κώδικα και βοηθά στην αποφυγή σφαλμάτων μέσω του ελέγχου τύπων.



## Κελυφοποίηση και απόκρυψη πληροφοριών

Σε αρκετές σύγχρονες γλώσσες προγραμματισμού δίνεται ιδιαίτερη έμφαση στις έννοιες της «κελυφοποίησης» (encapsulation) και της «απόκρυψης πληροφοριών» (information hiding), σύμφωνα με τις οποίες:

- Τα δεδομένα είναι στενά συνυφασμένα με τις λειτουργίες που επιτρέπονται πάνω σε αυτά και, ως εκ τούτου, είναι σκόπιμο να τοποθετούνται στο ίδιο «κέλυφος». Με άλλα λόγια, σε μια κοινή και σχετικά αυτοτελή δομική μονάδα στον πηγαίο κώδικα.
- Οι λεπτομέρειες της υλοποίησης μιας τέτοιας δομικής μονάδας δεν είναι συνήθως ενδιαφέρουσες παρά μόνο στο εσωτερικό της. Αντίθετα, είναι σκόπιμο να μπορεί να αντικατασταθεί η υλοποίηση μιας δομικής μονάδας χωρίς να επηρεάζεται το υπόλοιπο πρόγραμμα.

Ορισμένες γλώσσες προγραμματισμού υποστηρίζουν άμεσα τις έννοιες της «κελυφοποίησης» και της «απόκρυψης πληροφοριών», παρέχοντας κατάλληλους τύπους δεδομένων. Αυτό όμως δεν σημαίνει ότι δεν μπορούν να εφαρμοστούν σε άλλες γλώσσες προγραμματισμού με κατάλληλο σχεδιασμό και ευθύνη του προγραμματιστή.

## Προστακτικός, αντικειμενοστρεφής και συναρτησιακός προγραμματισμός

Οι πρώτες γλώσσες προγραμματισμού που αναπτύχθηκαν ήταν προστακτικές. Μέχρι σήμερα οι προστακτικές γλώσσες χρησιμοποιούνται ευρέως για την ανάπτυξη λογισμικού. Τις δυο τελευταίες όμως δεκαετίες δύο νέες προγραμματιστικές τεχνοτροπίες δείχνουν να κερδίζουν έδαφος.

Οι προστακτικές (imperative) γλώσσες υποστηρίζουν μεταβλητές διαφόρων τύπων, δομημένων και μη, αναθέσεις τιμών σε μεταβλητές και εντολές που αλλάζουν τη ροή του ελέγχου. Το προγραμματιστικό ύφος που προωθούν συμβαδίζει με αυτό της γλώσσας μηχανής, είναι όμως σημαντικά υψηλότερου επιπέδου. Προστακτικές γλώσσες που χρησιμοποιούνται κατά κόρον σήμερα στην ανάπτυξη λογισμικού είναι η C, η Pascal και η Basic, οι δυο τελευταίες συχνά επαυξημένες με νέα χαρακτηριστικά. Στο παρελθόν, ιδιαίτερα δημοφιλείς προστακτικές γλώσσες ήταν η Fortran και η Cobol.

Στον αντικειμενοστρεφή (object-oriented) προγραμματισμό ιδιαίτερη έμφαση δίνεται στις έννοιες του αντικειμένου, της κλάσης, της κληρονομικότητας και της κελυφοποίησης. Οι περισσότερες αντικειμενοστρεφείς γλώσσες βασίζονται στο προστακτικό ύφος προγραμματισμού. Σε μια προσπάθεια εισαγωγικής περιγραφής της ιδέας του αντικειμενοστρεφούς προγραμματισμού, τα αντικείμενα και οι κλάσεις αντικαθιστούν τις μεταβλητές και τους τύπους δεδομένων, ενώ η κληρονομικότητα επιτρέπει τον ορισμό κλάσεων που κληρονομούν ιδιότητες και συμπεριφορά από άλλες κλάσεις. Σύγχρονες δημοφιλείς αντικειμενοστρεφείς γλώσσες προγραμματισμού είναι η C++ και η Java. Παρατηρείται επίσης η τάση εισαγωγής αντικειμενοστρεφών χαρακτηριστικών σε άλλες γλώσσες, όπως στην Pascal και την Basic.

Κάποιοι ισχυρίζονται ότι ο συναρτησιακός (functional) προγραμματισμός είναι πλησιέστερος στον τρόπο σκέψης του ανθρώπου απ' ό,τι ο προστακτικός. Στις συναρτησιακές γλώσσες κύρια θέση κατέχουν οι έννοιες της συνάρτησης και της αναδρομής, ενώ οι μεταβλητές και οι αναθέσεις χρησιμοποιούνται ελάχιστα ή εκλείπουν εντελώς. Δημοφιλείς γλώσσες συναρτησιακού προγραμματισμού είναι οι Lisp και Scheme, και πιο πρόσφατα οι Standard ML και Haskell, οι οποίες διαθέτουν ισχυρά συστήματα τύπων. Σε σύγκριση με τις προστακτικές γλώσσες οι συναρτησιακές διαθέτουν συνήθως απλούστερη και κομψότερη σημασιολογία, είναι όμως μέχρι σήμερα σαφώς λιγότερο δημοφιλείς στην ανάπτυξη λογισμικού.

## **Πιθανές πηγές προβλημάτων**

Ορισμένα χαρακτηριστικά των γλωσσών καθιστούν τον προγραμματισμό πιο επιρρεπή σε σφάλματα. Αυτό, φυσικά, δεν σημαίνει ότι τα χαρακτηριστικά αυτά δεν είναι χρήσιμα ή ότι θα έπρεπε να εκλείπουν. Αντίθετα, αρκετές φορές είναι απαραίτητα και διευκολύνουν εξαιρετικά την ανάπτυξη λογισμικού. Σημαίνει όμως ότι οι προγραμματιστές οφείλουν να είναι ιδιαίτερα προσεκτικοί στη χρήση τους, η οποία πρέπει να γίνεται με μέτρο και πειθαρχία. Τέτοια χαρακτηριστικά είναι τα ακόλουθα:

- *Εντολή «goto», εξαιρέσεις, διακοπές.* Οι εντολές μεταφοράς της ροής ελέγχου που δεν ακολουθούν τη συντακτική δομή βλάπτουν συνήθως την αναγνωσιμότητα του πηγαίου κώδικα. Η εντολή «goto» μπορεί να αποφευχθεί συνήθως με τη χρήση ειδικών εντολών άμεσης εξόδου από βρόχους και εξαιρέσεων που περιγράφηκαν αναλυτικά στην προηγούμενη παράγραφο.
- *Αριθμοί κινητής υποδιαστολής.* Προβλήματα παρουσιάζονται λόγω της εγγενούς ανακρίβειας των αριθμών κινητής υποδιαστολής, που αποτελεί αντικείμενο μελέτης του κλάδου της αριθμητικής ανάλυσης. Ιδιαίτερη προσοχή χρειάζεται στη σύγκριση ισότητας ή ανισότητας τέτοιων αριθμών, καθώς και για να αποφευχθεί η συσσώρευση αριθμητικών σφαλμάτων.
- *Μετατροπές τύπων.* Οι περισσότερες γλώσσες επιτρέπουν τη μετατροπή δεδομένων από έναν τύπο σε κάποιον άλλο είτε ρητά με μέριμνα του προγραμματιστή είτε και με αυτόματο τρόπο με μέριμνα της υλοποίησης της γλώσσας. Οι μετατροπές τύπων συχνά οδηγούν σε απώλεια πληροφορίας ή ακόμα και σε παραποίηση της, όταν ο τελικός τύπος δεδομένων δεν είναι σε θέση να αναπαραστήσει την αρχική τιμή.
- *Δείκτες και δυναμική παραχώρηση μνήμης.* Οι δείκτες είναι χαρακτηριστικό προγραμματισμού χαμηλού επιπέδου, που διατηρείται όμως σε πολλές σύγχρονες γλώσσες προγραμματισμού. Ο κίνδυνος που εμπεριέχει η χρήση τους προέρχεται κυρίως από το γεγονός ότι αντικείμενα της μνήμης είναι δυνατό να προσπελαύνονται από περισσότερα ονόματα (aliasing). Η απουσία αρχικοποίησης των δεικτών, καθώς και η μη απελευθέρωση μνήμης που έχει παραχωρηθεί δυναμικά είναι υπεύθυνες για ένα μεγάλο αριθμό σφαλμάτων που συνήθως είναι εξαιρετικά δύσκολο να εντοπιστούν.
- *Αναδρομικές κλήσεις.* Αναδρομή είναι η κλήση μιας συνάρτησης από τον εαυτό της είτε απευθείας είτε μέσω κλήσεων σε άλλες συναρτήσεις. Η χρήση της οδηγεί συνήθως σε ιδιαίτερα κομψά και μικρά προγράμματα σε σύγκριση με τη χρήση βρόχων. Όμως, σε κάποιες περιπτώσεις καθιστά δυσκολότερη την κατανόηση του

προγράμματος και είναι πιο επιρρεπής στον κίνδυνο μη τερματισμού ή εξάντλησης της μνήμης.

- *Παράλληλη επεξεργασία.* Ο παραλληλισμός είναι επικίνδυνος ως προγραμματιστική τεχνική, γιατί είναι δύσκολο να προβλεφθεί η αλληλεπίδραση μεταξύ των διαφορετικών διεργασιών που τρέχουν παράλληλα. Οι γλώσσες προγραμματισμού που τον υποστηρίζουν θα πρέπει να παρέχουν τη δυνατότητα συγχρονισμού και αλληλοαποκλεισμού των διεργασιών, ώστε η αλληλεπίδραση να μην είναι επιβλαβής και να μην καταλήγει σε αδιέξοδα.

## ΕΝΟΤΗΤΑ 6.4. ΤΕΧΝΙΚΕΣ ΣΥΓΓΡΑΦΗΣ ΠΗΓΑΙΟΥ ΚΩΔΙΚΑ

Δεν υπάρχει η «χρυσή βίβλος» που να περιέχει συμβουλές για τη συγγραφή αλάνθαστου πηγαίου κώδικα, διότι, πολύ απλά, δεν υπάρχει αλάνθαστος πηγαίος κώδικας. Μπορεί, ωστόσο, κανείς να λάβει ορισμένα μέτρα, προκειμένου να συγγράφει όσο το δυνατόν καλύτερο κώδικα. Η περιγραφή τέτοιων μέτρων με λεπτομέρεια δεν είναι δυνατή εφόσον στο παρόν βιβλίο δεν αναφερόμαστε σε κάποια συγκεκριμένη γλώσσα προγραμματισμού. Παρ' όλα αυτά, είναι χρήσιμη η παράθεση μιας σειράς συμβουλών για τη συγγραφή πηγαίου κώδικα. Με το θέμα ασχολείται η παρούσα ενότητα.

### 6.4.1. Τεχνικές αποφυγής σφαλμάτων

Η αποφυγή σφαλμάτων (fault avoidance) αποσκοπεί στη συγγραφή πηγαίου κώδικα με όσο το δυνατό λιγότερα σφάλματα. Η πιστή χρήση τεχνικών αποφυγής σφαλμάτων είναι ο ακρογωνιαίος λίθος στην ανάπτυξη αξιόπιστου λογισμικού. Δεδομένου του πολύ υψηλού κόστους εντοπισμού και διόρθωσης σφαλμάτων στον πηγαίο κώδικα, η διαδικασία ανάπτυξης λογισμικού πρέπει να είναι προσανατολισμένη κυρίως προς την αποφυγή των λαθών κατά τη συγγραφή του. Κατάλληλα εργαλεία και τεχνικές είναι διαθέσιμα για αυτό τον σκοπό. Γενικά, η συγγραφή κώδικα χωρίς σφάλματα διευκολύνεται από τη σωστή προδιαγραφή των απαιτήσεων, από την αξιοποίηση των

χαρακτηριστικών της γλώσσας προγραμματισμού και, ασφαλώς, από τον ίδιο τον ανθρώπινο παράγοντα.

### **Προδιαγραφή απαιτήσεων**

Η ύπαρξη ενός αυστηρού τρόπου περιγραφής της προδιαγραφής των απαιτήσεων από το λογισμικό διευκολύνει εξαιρετικά την αποφυγή σφαλμάτων, καθώς αποφεύγονται παρερμηνείες στην επιθυμητή λειτουργία του πηγαίου κώδικα. Η πειθαρχημένη εφαρμογή των προβλεπόμενων μέχρι την κωδικοποίηση ενεργειών ανάπτυξης λογισμικού αποτελεί ισχυρό εργαλείο αποφυγής σφαλμάτων. Η σαφήνεια στην περιγραφή όλων των ενδιάμεσων προϊόντων (προδιαγραφές, σχεδίαση) και η αποφυγή «ευκόλως εννοούμενων» συνήθως αποδίδουν τον χρόνο και τον κόπο που απαιτούν. Επιπλέον, έχουν αναπτυχθεί και τυπικές μέθοδοι (formal methods) για να γίνεται έλεγχος, χειρωνακτικά ή και ενίοτε αυτόματα, αν ο πηγαίος κώδικας που έχει γραφεί πληροί τις απαιτήσεις. Από την άλλη πλευρά όμως, το κόστος συγγραφής απαιτήσεων είναι συνήθως υψηλότερο όταν η γλώσσα που χρησιμοποιείται είναι αυστηρότερη ή τυπική.

### **Αξιοποίηση της γλώσσας προγραμματισμού**

Η πολιτική συγγραφής πηγαίου κώδικα πρέπει να αποβλέπει στην αξιοποίηση των θετικών χαρακτηριστικών μιας γλώσσας προγραμματισμού και στην αποφυγή χρήσης αρνητικών χαρακτηριστικών της. Για παράδειγμα, αν μια γλώσσα προγραμματισμού δεν είναι ισχυρή στους μαθηματικούς υπολογισμούς, τότε δεν είναι καλή ιδέα να χρησιμοποιηθεί για τη συγγραφή εφαρμογής στην οποία οι μαθηματικού υπολογισμοί έχουν ιδιαίτερη βαρύτητα.

### **Επιδίωξη ποιότητας**

Η ομάδα κωδικοποίησης πρέπει να διαθέτει αυξημένη ευαισθησία για την εξασφάλιση της απαιτούμενης ποιότητας και την ανάπτυξη πηγαίου κώδικα που να διαθέτει τα επιθυμητά χαρακτηριστικά της Ενότητας 5.2. Οι προγραμματιστές πρέπει να εφαρμόζουν αυστηρά τη συνολική πολιτική συγγραφής κώδικα και να επιδιώκουν εξ αρχής την ανάπτυξη κώδικα χωρίς σφάλματα.



#### 6.4.2. Ανοχή σε σφάλματα

**Ανοχή σε σφάλματα** (fault tolerance). Δεδομένης της πολυπλοκότητας των σύγχρονων συστημάτων λογισμικού, τις περισσότερες φορές δεν είναι εφικτή η αποφυγή ή ο εντοπισμός των σφαλμάτων. Ακόμα κι αν αυτά ήταν εφικτά, το κόστος για να επιτευχθούν είναι συνήθως απαγορευτικό. Για τον λόγο αυτό, συστήματα λογισμικού με αυξημένες απαιτήσεις αξιοπιστίας οφείλουν να προβλέπουν την πιθανή εμφάνιση σφαλμάτων στη λειτουργία τους και να δείχνουν σχετική ανοχή. Ειδικές τεχνικές χρησιμοποιούνται για τον σκοπό αυτό.

Συνήθως απαιτείται από τα συστήματα λογισμικού να συνεχίζουν τη λειτουργία τους μετά την εμφάνιση κάποιων σφαλμάτων ή άλλων απροσδόκητων καταστάσεων με τις μικρότερες δυνατές απώλειες. Σε ορισμένες περιπτώσεις, για παράδειγμα στο λογισμικό που χρησιμοποιείται στον υπολογιστή ενός αεροσκάφους εν πτήση ή για τον έλεγχο εναέριας κυκλοφορίας, η συνολική αποτυχία ενός συστήματος λογισμικού θα είχε καταστροφικά αποτελέσματα και πρέπει να αποφευχθεί πάση θυσία. Για τον λόγο αυτό, και δεδομένης της αδυναμίας εξασφάλισης του αλάθητου των συστημάτων λογισμικού με τις τεχνικές αποφυγής και εντοπισμού σφαλμάτων, στην ανάπτυξη συστημάτων λογισμικού υψηλού κινδύνου γίνεται πρόβλεψη για την προφύλαξη από σφάλματα που ενδέχεται να παρουσιαστούν μετά την έναρξη λειτουργίας τους.

Για την ανάπτυξη συστημάτων λογισμικού ανεκτικών σε σφάλματα απαιτείται η προσθήκη επιπλέον κώδικα που αναλαμβάνει τις ακόλουθες λειτουργίες:

- **Δυναμικός εντοπισμός σφαλμάτων.** Το σύστημα λογισμικού πρέπει να είναι σε θέση να διαγνώσει ότι παρουσιάστηκε σφάλμα κατά τη λειτουργία του. Η διάγνωση τέτοιου είδους σφαλμάτων είναι απαραίτητο να γίνεται όσο το δυνατό νωρίτερα, ώστε να μη διευρύνονται και να μην επηρεάζουν ανεξάρτητα τμήματα του συστήματος που λειτουργούν σωστά.
- **Ανάνηψη από σφάλματα.** Σε περίπτωση εντοπισμού σφάλματος κατά τη λειτουργία του, το σύστημα λογισμικού πρέπει να επιδιορθώσει τη βλάβη και να αποκαταστήσει τη φυσιολογική του

λειτουργία. Αυτό μπορεί να επιτευχθεί είτε με την οπισθοχώρηση σε μια προηγούμενη ασφαλή κατάσταση του συστήματος είτε με τη διόρθωση της βλάβης και τη μετάβαση σε μια νέα, σωστή κατάσταση. Στην περίπτωση συστημάτων λογισμικού που διαχειρίζονται βάσεις δεδομένων χρησιμοποιείται συνήθως η τακτική οπισθοχώρησης και συγκεκριμένα η τεχνική των συναλλαγών (transactions). Αν κάποια συναλλαγή δεν είναι δυνατό να ολοκληρωθεί λόγω της εμφάνισης σφάλματος, τότε ολόκληρη η συναλλαγή ακυρώνεται, προκειμένου να επανέλθει το σύστημα σε μια ασφαλή και συνεπή κατάσταση. Η τακτική μετάβασης σε νέα, σωστή κατάσταση είναι συνήθως δυσκολότερη στην υλοποίηση.

- **Πρόληψη μελλοντικών σφαλμάτων.** Η ανάνηψη από ένα σφάλμα που παρουσιάστηκε κατά τη διάρκεια της λειτουργίας ενός συστήματος λογισμικού δεν είναι φυσικά αρκετή. Πρέπει να ληφθεί πρόνοια ώστε το σφάλμα να μην επαναληφθεί στο μέλλον. Όταν αυτό δεν είναι δυνατό να επιτευχθεί από το ίδιο το σύστημα λογισμικού με την αυτόματη επιδιόρθωση του τμήματος που προκάλεσε το σφάλμα, απαιτείται η επέμβαση του κατασκευαστή. Στην περίπτωση αυτή το σύστημα οφείλει να γνωστοποιήσει την ύπαρξη του σφάλματος και να επιτρέπει τη συντήρησή του με το ελάχιστο δυνατό κόστος.

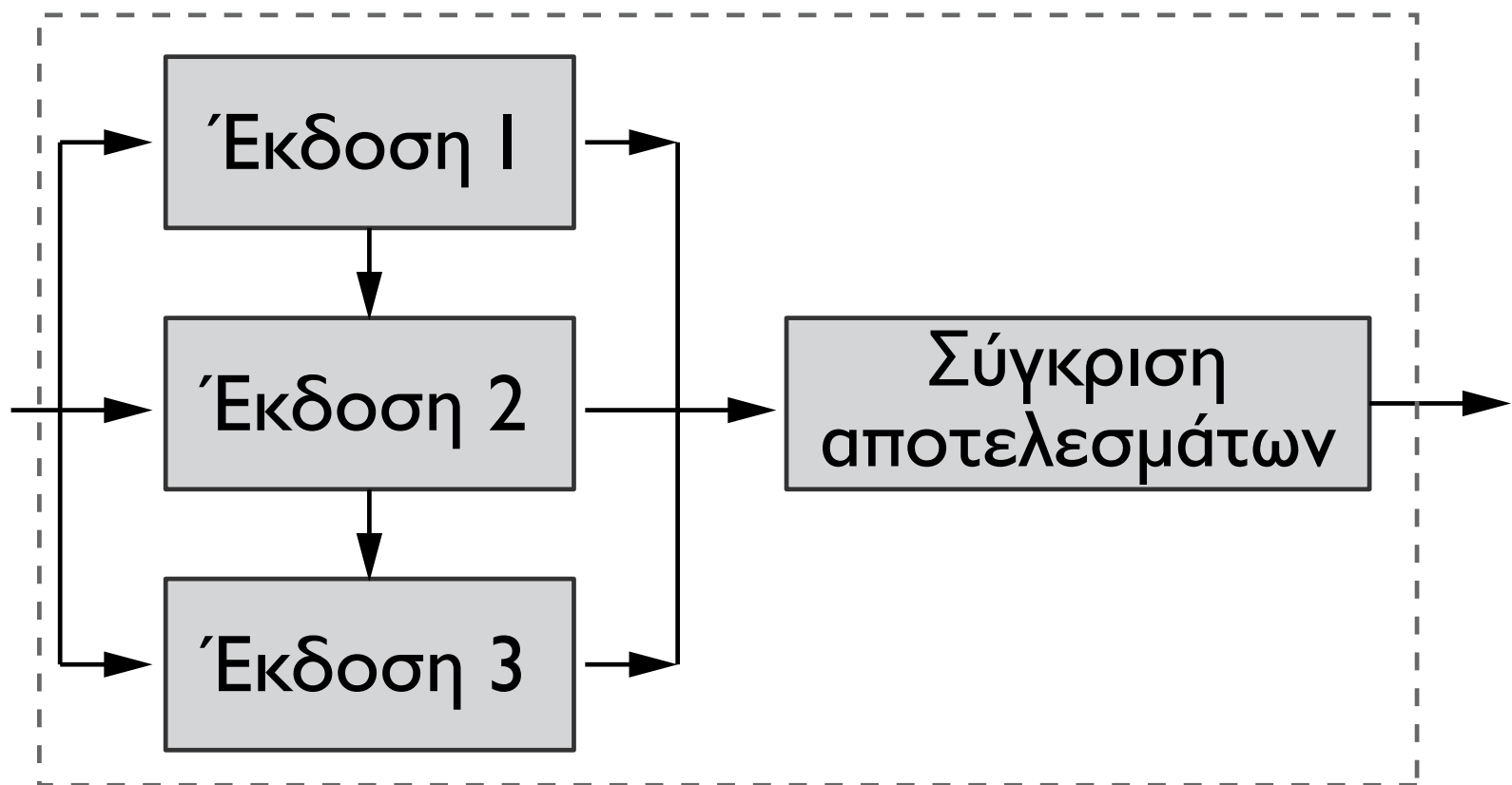
Διάφορες τεχνικές έχουν προταθεί για την υλοποίηση συστημάτων ανεκτικών σε σφάλματα. Δυο από αυτές περιγράφονται στη συνέχεια.

**Προγραμματισμός πολλών εκδόσεων.** Σύμφωνα με αυτή την τεχνική, το σύστημα λογισμικού υλοποιείται σε έναν αριθμό πλήρως λειτουργικών διαφορετικών εκδόσεων που αναπτύσσονται από διαφορετικές ομάδες. Οι εκδόσεις αυτές εκτελούνται παράλληλα με τα ίδια δεδομένα εισόδου και τα αποτελέσματά τους συγκρίνονται μεταξύ τους. Στην περίπτωση ασυμφωνίας των αποτελεσμάτων εφαρμόζεται ο κανόνας της πλειοψηφίας και οι εκδόσεις που οδήγησαν σε διαφορετικά αποτελέσματα απομονώνονται ως ελαττωματικές. Είναι φανερό ότι η τεχνική αυτή έχει σημαντικό κόστος, ωστόσο σε λογισμικό που χρησιμοποιείται σε κρίσιμες εφαρμογές συνήθως δικαιολογείται το μεγάλο κόστος. Τουλάχιστον τρεις εκδόσεις του

συστήματος πρέπει να είναι διαθέσιμες, όπως φαίνεται στο Σχήμα 6.4. Μια παραλλαγή είναι η τεχνική να περιοριστεί μόνο στα επισφαλή τμήματα του λογισμικού, οπότε μειώνεται και το κόστος.



**Σχήμα 6.4** Τεχνική τριών εκδόσεων.



**Αμυντικός προγραμματισμός** (defensive programming). Σύμφωνα με αυτή την τεχνική, οι προγραμματιστές ενσωματώνουν στον πηγαίο κώδικα ελέγχους για τη συνέπεια των δεδομένων και την καλή λειτουργία, υιοθετώντας μια αμυντική στάση και προβλέποντας την περίπτωση ασυνεπειών. Αν κάποιος έλεγχος αποτύχει, τότε θεωρείται ότι εντοπίστηκε σφάλμα και εφαρμόζονται κατάλληλες τεχνικές ανάνηψης και μελλοντικής πρόληψης.

#### 6.4.3. Εντοπισμός και διόρθωση σφαλμάτων

Ο εντοπισμός και η διόρθωση των σφαλμάτων που υπάρχουν στο λογισμικό αποσκοπεί στην εύρεση σφαλμάτων που υπάρχουν στον πηγαίο κώδικα πριν από την παράδοση αυτού σε τελική χρήση. Οι εργασίες αυτές γίνονται κατά τη φάση του ελέγχου του λογισμικού.

### ΕΝΟΤΗΤΑ 6.5. ΕΠΑΝΑΧΡΗΣΙΜΟΠΟΙΗΣΗ ΜΟΝΑΔΩΝ ΠΡΟΓΡΑΜΜΑΤΟΣ

Επαναχρησιμοποίηση (reuse) είναι η διαδικασία της υλοποίησης νέου λογισμικού χρησιμοποιώντας συστατικά από ήδη κατασκευασμένο λογισμικό. Εκτός από τον πηγαίο κώδικα και τα εκτελέσιμα προγράμματα, επαναχρησιμοποιήσιμα συστατικά είναι επίσης τα αποτελέσματα της ανάλυσης και της σχεδίασης, η τεκμηρίωση, δεδομένα και σχέδια ελέγχου κ.λπ. Η επαναχρησιμοποίηση είναι βασική αρχή σε κάθε επιστήμη που σχετίζεται με την ανάπτυξη και την παραγωγή. Αποτελεί βάση για τη δραστική βελτίωση της ποιότητας και της αξιοπιστίας του λογισμικού, καθώς και για τη μακροπρόθεσμη μείωση του κόστους ανάπτυξής του.

Προκειμένου να είναι επαναχρησιμοποιήσιμος, ο πηγαίος κώδικας πρέπει να έχει γραφεί όχι μόνο για τη στενή του χρήση στο πλαίσιο κάποιας συγκεκριμένης εφαρμογής αλλά με την πρόβλεψη να μπορεί να χρησιμοποιηθεί στο μέλλον σε παρόμοιες περιπτώσεις. Η γενικότητα εξασφαλίζεται με την παραμετροποίηση, όπου αυτό είναι δυνατό, σε βάρος συχνά του αρχικού

κόστους ανάπτυξης του πηγαίου κώδικα. Επίσης, ο πηγαίος κώδικας πρέπει να είναι μεταφέρσιμος, να έχει τη μικρότερη δυνατή αλληλεπίδραση με τον υπόλοιπο κώδικα και, όπου αυτή η αλληλεπίδραση είναι αναπόφευκτη, να είναι καλά τεκμηριωμένη.

Η επαναχρησιμοποίηση έχει για πολύ καιρό αποτελέσει αντικείμενο συζητήσεων στην κοινότητα των κατασκευαστών και των ερευνητών. Παρά τις κατά καιρούς πομπώδεις δηλώσεις περί μεθοδολογιών ή εργαλείων που εξασφαλίζουν την επαναχρησιμοποίηση, κάτι τέτοιο δεν έχει γίνει μέχρι σήμερα δυνατό σε μεγάλη κλίμακα. Τέτοιες δηλώσεις συνόδευσαν επί μακρόν τις αντικειμενοστρεφείς μεθοδολογίες ανάπτυξης λογισμικού, ωστόσο το ενδιαφέρον είναι ότι μέχρι σήμερα η επαναχρησιμοποίηση είναι πραγματικότητα ίσως περισσότερο για γλώσσες προγραμματισμού που χρησιμοποιούνται σε μαθηματικούς υπολογισμούς με τη μορφή βιβλιοθηκών συναρτήσεων και λιγότερο για γενικής χρήσης λογισμικό. Η τάση είναι ασφαλώς προς την επικράτηση της επαναχρησιμοποίησης ως τεχνικής, και ακόμη και μέσα στον ίδιο κατασκευαστή λογισμικού η επαναχρησιμοποίηση είναι ανάγκη, όχι απλά επίτευγμα.

## ΕΝΟΤΗΤΑ 6.6. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ

### 6.6.1. Δραστηριότητες

#### Δραστηριότητα I/Κεφάλαιο 6

---

Η απάντηση είναι απλά «όχι». Στην πράξη, η σχεδίαση είναι σπάνια τόσο λεπτομερής. Η σχεδίαση συνήθως στέκεται στις κατευθυντήριες γραμμές για τη συγγραφή πηγαίου κώδικα και δεν ασχολείται με τις λεπτομέρειές της. Η χρήση προγραμματιστικών τεχνικών και άλλοι πρακτικοί λόγοι στην κατασκευή προγραμμάτων αλλά και τα ιδιαίτερα χαρακτηριστικά της γλώσσας προγραμματισμού που χρησιμοποιείται είναι οι σημαντικότερες αιτίες για

τις οποίες ο προγραμματιστής κάθε άλλο παρά εύκολη και ανιαρή δουλειά έχει να κάνει.

## Δραστηριότητα 2/Κεφάλαιο 6

---

Δύο πολύ σημαντικοί λόγοι είναι: α) η δυνατότητα κατανόησης του πηγαίου κώδικα που έχει συγγράψει κάποιος και από έναν άλλο προγραμματιστή και β) η δυνατότητα κατανόησης του κώδικα από τον ίδιο τον συγγραφέα του μετά από λίγο καιρό.

Μπορείτε να κάνετε μια μικρή έρευνα και θα διαπιστώσετε ότι οι προγραμματιστές συχνά αρνούνται να πειράξουν κώδικα που δεν είναι δικός τους, αλλά και συχνά δυσκολεύονται να κατανοήσουν κώδικα που έγραψαν παλιά. Άλλοι λόγοι που μπορούν να αναφερθούν εδώ είναι η δυνατότητα ταξινόμησης, χαρακτηρισμού και, γενικά, διαχείρισης του πηγαίου κώδικα από ειδικά εργαλεία που υπάρχουν για τον σκοπό αυτό (όπως τα εργαλεία reverse engineering).

### 6.6.2. Ασκήσεις αυτοαξιολόγησης

#### Άσκηση I/Κεφάλαιο 6

---

Οι σημαντικότερες κατηγορίες εργαλείων που χρησιμοποιεί ο προγραμματιστής είναι οι ακόλουθες:

- Συντάκτες προγραμμάτων (program editors)
- Υλοποιήσεις γλωσσών προγραμματισμού
- Εντοπιστές σφαλμάτων (debuggers)
- Γεννήτορες προγραμμάτων (program generators)
- Ολοκληρωμένα περιβάλλοντα προγραμματισμού

Μπορείτε να ανατρέξετε στο περιβάλλον σας και να διαπιστώσετε ότι για το μεγαλύτερο διάστημα οι προγραμματιστές αρκούσαν στα δύο πρώτα στην καλύτερη περίπτωση και σε ένα εργαλείο της τρίτης κατηγορίας. Τα

σύγχρονα και ολοκληρωμένα περιβάλλοντα που γνώρισαν οι νεότεροι είναι σχετικά πρόσφατα. Η εντύπωση που αποκτά κανείς διερευνώντας το θέμα είναι αντίστοιχη με αυτή του τσαγκάρη που κατασκευάζει το παπούτσι με τη φαλτσέτα, το σφυρί και το καρφί, από τη μία, και της σύγχρονης υποδηματοβιομηχανίας, από την άλλη.

## Άσκηση 2/Κεφάλαιο 6

---

Τα επιθυμητά χαρακτηριστικά του πηγαίου κώδικα μπορούν να συνοψιστούν στα ακόλουθα:

**Επάρκεια**, δηλαδή λειτουργία του λογισμικού χωρίς σφάλματα και με σωστή χρήση της υπολογιστικής μηχανής που το στεγάζει.

**Επιδόσεις**, δηλαδή μεγάλη ταχύτητα εκτέλεσης του πηγαίου κώδικα.

**Αναγνωσιμότητα**, δηλαδή δυνατότητα κατανόησης του πηγαίου κώδικα και από άλλους προγραμματιστές χωρίς κόπο.

**Τεκμηρίωση**, δηλαδή συνοδεία του πηγαίου κώδικα με έγγραφα που συμβάλλουν στην κατανόησή του.

**Μεταφερσιμότητα**, δηλαδή δυνατότητα του πηγαίου κώδικα να μπορεί να εκτελεστεί χωρίς αλλαγές σε διαφορετικά περιβάλλοντα (λειτουργικά συστήματα, υπολογιστές) στα οποία διατίθενται εκδόσεις της γλώσσας προγραμματισμού που χρησιμοποιείται.

**Δυνατότητα επαναχρησιμοποίησης**, δηλαδή χρήσης του πηγαίου κώδικα και σε άλλα συστήματα λογισμικού που θα κατασκευαστούν στο μέλλον.

Αν και είναι γνωστά τα χαρακτηριστικά που θέλουμε να έχει ο πηγαίος κώδικας, δεν είναι πάντα εύκολη η συγγραφή προγραμμάτων που τα ενσωματώνουν. Συνήθως, πολλά από τα χαρακτηριστικά αυτά θεωρούνται περιττά όπως, για παράδειγμα, η συγγραφή επαρκών σχολίων. Ο αναγνώστης ενθαρρύνεται να μην υποτιμήσει κανένα από αυτά τα χαρακτηριστικά και να προσπαθεί να γράφει πηγαίο κώδικα που τα ενσωματώνει. Μπορεί στις εκπαιδευτικές του εργασίες να μη φαίνεται η σημασία τους, στην πράξη όμως δεν συμβαίνει το ίδιο και είναι καλό να έχει συνηθίσει κανείς να γράφει καλό πηγαίο κώδικα από νωρίς.

## Άσκηση 3/Κεφάλαιο 6

---

Κάθε αλγόριθμος είναι δυνατό να κωδικοποιηθεί χρησιμοποιώντας τρία μόνο στοιχεία ελέγχου, τα οποία είναι:

- Η ακολουθία (`s1; s2`), δηλαδή η σειριακή εκτέλεση εντολών.
- Η επιλογή (`if c then s1 else s2`), δηλαδή η επιλεκτική εκτέλεση μιας εντολής αν ισχύει μια λογική συνθήκη είτε μιας άλλης εντολής αν η ίδια λογική συνθήκη δεν ισχύει.
- Η επανάληψη (`while c do s`), δηλαδή η επαναληπτική εκτέλεση μιας εντολής για όσο διάστημα παραμένει αληθής μια λογική συνθήκη.

Τα στοιχεία αυτά αποτελούν τη βάση του δομημένου προγραμματισμού. Με απλά λόγια, κάθε μονάδα δομημένου προγράμματος έχει ένα και μόνο σημείου εισόδου και ένα σημείου εξόδου. Όσο αυτονόητο και αν φαίνεται αυτό σήμερα, στο παρελθόν η ανεξέλεγκτη χρήση της εντολής «`goto`» είχε δημιουργήσει μια χαοτική κατάσταση κατά την οποία τα σημεία εισόδου και εξόδου από μονάδες προγράμματος ήταν πολλά και τελικά ανεξέλεγκτα. Το φαινόμενο ονομάστηκε «*spaghetti programming*».

## ΒΙΒΛΙΟΓΡΑΦΙΑ

Pressman, R. S., *Software Engineering-A Practitioners Approach*, McGraw-Hill.

Sommerville, I. *Software Engineering*, London: Addison-Wesley.

## ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗΣ ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ

Σκοπός του κεφαλαίου είναι η εισαγωγή του αναγνώστη στην αντικειμενοστρεφή τεχνολογία και γενικότερα στα σύγχρονα χαρακτηριστικά και τις τρέχουσες τάσεις στην ανάπτυξη του λογισμικού. Επίσης, θα παρουσιάσουν οι βασικές έννοιες που χρησιμοποιούνται στην αντικειμενοστρεφή τεχνολογία και η ορολογία και τα βασικά στοιχεία των συμβολισμών που θα χρησιμοποιηθούν στη συνέχεια.

Μετά τη μελέτη του κεφαλαίου αυτού, ο αναγνώστης θα είναι σε θέση:

- να αναφέρει σύγχρονα χαρακτηριστικά του λογισμικού και του τρόπου ανάπτυξής του,
- να αναφέρει αδυναμίες της δομημένης φιλοσοφίας στην αντιμετώπιση των οποίων μπορεί να συμβάλλει η αντικειμενοστρεφής τεχνολογία,
- να ορίζει τις βασικές έννοιες της αντικειμενοστρεφούς τεχνολογίας,
- να ορίζει τρεις συσχετίσεις μεταξύ κλάσεων, οι οποίες χρησιμοποιούνται στην αντικειμενοστρεφή τεχνολογία,
- να αναγνωρίζει τους βασικούς συμβολισμούς της UML για την παράσταση κλάσεων και συσχετίσεων μεταξύ τους.

### Έννοιες-κλειδιά

---

- Αντικειμενοστρεφής τεχνολογία
- Κλάση
- Αντικείμενο
- Πεδίο
- Μέθοδος
- Ιεραρχία



- Αφαίρεση
- Συσχέτιση
- Κληρονομικότητα
- Συναρμολόγηση
- Στιγμιότυπο
- UML

## Σύνοψη

---

Η δομημένη ανάλυση και σχεδίαση κυριάρχησε για πολλά χρόνια στη θεωρία και την πρακτική της ανάπτυξης λογισμικού και ακόμη και σήμερα μπορούμε να θεωρούμε ότι πολλές από τις εφαρμογές που λειτουργούν έχουν κατασκευαστεί σύμφωνα με τη φιλοσοφία της. Τα συσσωρευμένα προβλήματα και οι εξελίξεις στον τομέα των υπολογιστών αλλά και η εξάπλωση του διαδικτύου και των εφαρμογών του δημιούργησαν μια νέα πραγματικότητα για την Τεχνολογία Λογισμικού. Η προσέγγιση που είναι κυρίαρχη σήμερα αναφέρεται ως «αντικειμενοστρεφής» και αποτελεί υπερσύνολο της δομημένης ανάλυσης και σχεδίασης.

Η αντικειμενοστρεφής φιλοσοφία μάς προσφέρει έναν νέο τρόπο να αναλύουμε τα προβλήματα του πραγματικού κόσμου σε συστατικά λογισμικού. Για το σκοπό αυτό εισάγει αρκετές νέες έννοιες και συμβολισμούς σχετικά με τις οποίες υπήρχε σύγχυση και πλουραλισμός απόψεων για πολλά χρόνια. Οι σημαντικότερες από αυτές είναι οι έννοιες «κλάση», «αντικείμενο», «συσχέτιση», «κληρονομικότητα», «γενίκευση» και «συναρμολόγηση». Το ζητούμενο είναι, πλέον, να προσπαθούμε να αναλύουμε με τέτοιους όρους τα συστήματα λογισμικού που κατασκευάζουμε. Η αντικειμενοστρεφής φιλοσοφία δεν είναι με καμία έννοια πανάκεια, είναι ωστόσο το καλύτερο από τα εργαλεία που έχουμε σήμερα στα χέρια μας για να αντιμετωπίσουμε τα προβλήματα στην ανάπτυξη του λογισμικού.

Τα πρώτα βήματα της Τεχνολογίας Λογισμικού έγιναν μέσα σε ένα περιβάλλον όπου ο ρυθμός των αλλαγών που συντελούνταν στην τεχνολογία των υπολογιστών αλλά και σε όλους τους τομείς των ανθρώπινων δραστηριοτήτων όπου αυτοί χρησιμοποιούνται δεν ήταν συνειδητός, σίγουρα όχι στο βαθμό που είναι σήμερα. Οι πρωτοπόροι στην έρευνα και στην πρακτική της ανάπτυξης λογισμικού αναζητούσαν τον καλύτερο τρόπο να κατασκευάζεται και να συντηρείται λογισμικό χωρίς να γνωρίζουν ότι η έννοια «καλύτερο» ήταν μόνο σχετική, χωρίς να μπορούν να φανταστούν πόσο γρήγορα θα μεταβαλλόταν η άποψη, οι απαιτήσεις και, τελικά, η συνείδηση των χρηστών των υπολογιστικών συστημάτων, αλλά και χωρίς να μπορούν να συλλάβουν ποια μορφή θα έχουν οι υπολογιστές στο μέλλον.

Σήμερα γνωρίζουμε ότι οι υπολογιστές και, κατά συνέπεια, το λογισμικό βρίσκονται πρακτικά παντού και μας υποστηρίζουν σε πολλές από τις δραστηριότητές μας. Η αίσθηση και η μορφή του λογισμικού έχει εξελιχθεί. Η αλληλεπίδραση με το λογισμικό, την οποία βιώνουμε σήμερα, δεν γίνεται με λίγες και μεγάλες εφαρμογές, αλλά με πολλές και μικρές, οι οποίες βρίσκονται κατανεμημένες οπουδήποτε σε κάποιο δίκτυο, όπου πρόκειται να συγκλίνουν στο κοντινό μέλλον όλα τα πληροφοριακά και επικοινωνιακά συστήματα. Ως εκ τούτου, ο τρόπος και η φιλοσοφία της ανάπτυξης λογισμικού ήταν αναγκαίο να εξελιχθεί, ώστε να ικανοποιούνται οι σύγχρονες απαιτήσεις.

Σήμερα, πλέον, μιλάμε για κλάσεις και συνεργασία, ενώ στη δομημένη προσέγγιση μιλάμε για συναρτήσεις και δεδομένα. Οι πρώτες γλώσσες προγραμματισμού σκόπευαν στο να κάνουν περισσότερο προσιτό στον άνθρωπο τον προγραμματισμό των υπολογιστών απ' ό,τι επέτρεπε η γλώσσα μηχανής. Με τις δομικές μονάδες που προσέφεραν αυτές οι γλώσσες έπρεπε να χτιστούν οι οσοσδήποτε σύνθετες εφαρμογές λογισμικού. Μιλώντας μεταφορικά, ήταν τα υλικά που σε μεγάλο βαθμό καθόριζαν τον τρόπο σκέψης του χτίστη και όχι η σύλληψή του για το οικοδόμημα. Αυτό σταδιακά άλλαξε, και έτσι οι γλώσσες προγραμματισμού αλλά και ο τρόπος σκέψης όσων ασχολούνταν με την κατασκευή λογισμικού απέκτησαν στοιχεία δομής, τα οποία δεν είχε το μέσο άμεσου προγραμματισμού των υπολογιστών, δηλαδή η γλώσσα μηχανής. Η έννοια αυτή της δομής χαρακτήρισε τον τρόπο σκέψης μας στη σύλληψη και σχεδίαση του λογισμικού, η οποία χαρακτηρίστηκε «δομημένη».

Όμως και πάλι, ο τρόπος σκέψης είχε έντονα τα χαρακτηριστικά του εργαλείου (της όποιας γλώσσας προγραμματισμού) και, τουλάχιστον στην αρχή, καθοριζόταν από αυτό περισσότερο απ' ό,τι το καθόριζε. Η πολυπλοκότητα και η ανάγκη επεξεργασίας μεγάλου όγκου δεδομένων έφεραν στην επιφάνεια σημαντικές αδυναμίες της δομημένης φιλοσοφίας στην ανάπτυξη λογισμικού.

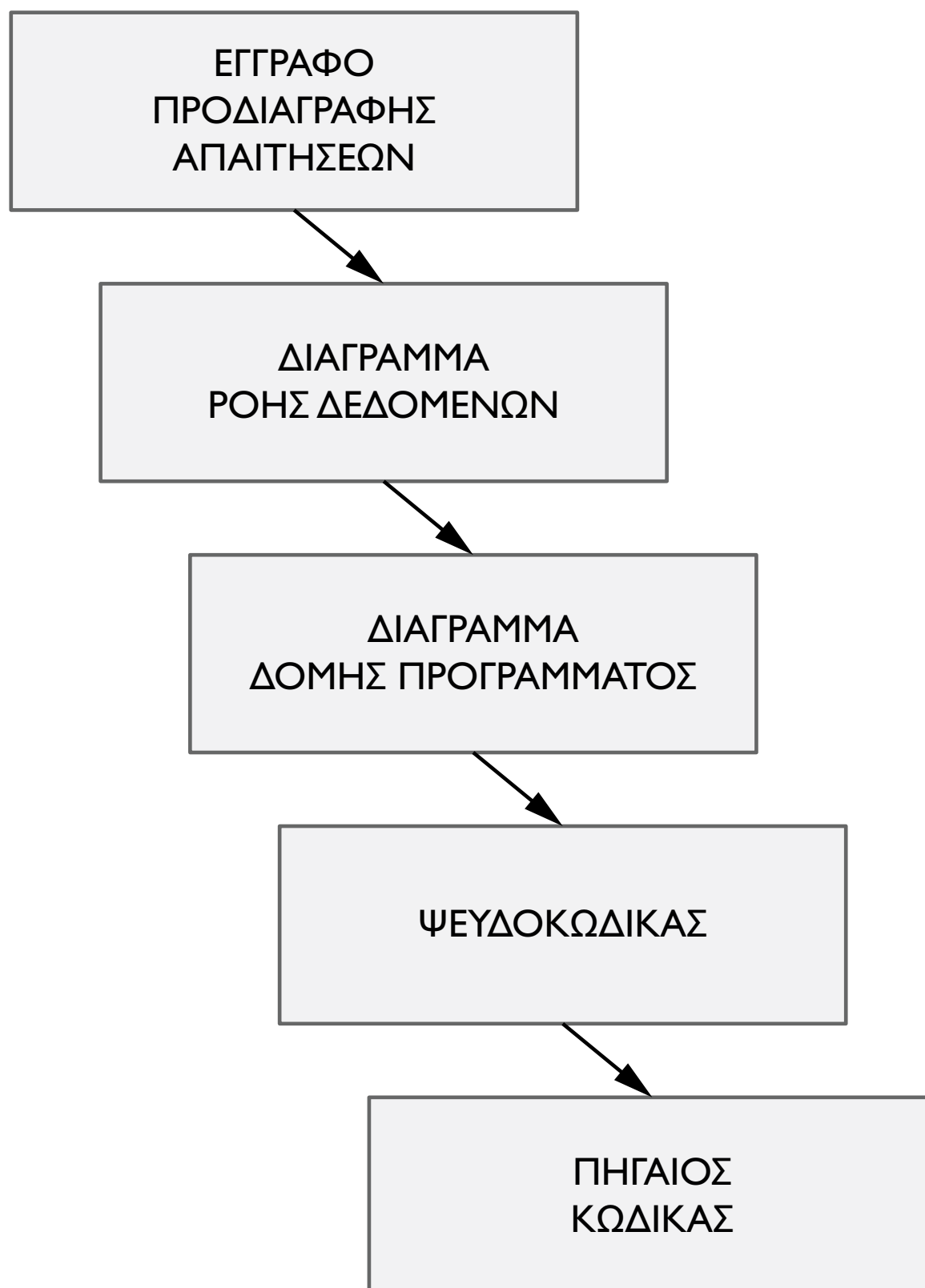
Το επόμενο βήμα, το οποίο μόνο πρόσφατα άρχισε να γίνεται συνειδητό, ήταν η ριζική μεταβολή του τρόπου σκέψης, έτσι ώστε –για να επανέλθουμε στη μεταφορά– να είναι τα εργαλεία του χτίστη που καθορίζονται από την άποψή του για το οικοδόμημα και όχι το αντίστροφο. Αυτή η θεώρηση επικράτησε να ονομάζεται «αντικειμενοστρεφής φιλοσοφία» ή «παράδειγμα» (*object-oriented philosophy, paradigm*) και αντιμετωπίζει με έναν εντελώς διαφορετικό από τη δομημένη προσέγγιση τρόπο τη σύλληψη, την ανάλυση και τη σχεδίαση του λογισμικού. Με τις βασικές έννοιες που ενσωματώνει η αντικειμενοστρεφής φιλοσοφία θα ασχοληθούμε στο κεφάλαιο αυτό.

## ΕΝΟΤΗΤΑ 7.1. Η ΠΟΡΕΙΑ ΤΗΣ ΔΟΜΗΜΕΝΗΣ ΑΝΑΛΥΣΗΣ ΚΑΙ ΣΧΕΔΙΑΣΗΣ

Στα πρώτα κεφάλαια του βιβλίου αυτού ασχοληθήκαμε με την οικογένεια προσεγγίσεων ανάπτυξης λογισμικού η οποία χαρακτηρίζεται ως δομημένη ανάλυση και σχεδίαση. Οι προσεγγίσεις αυτές εμφανίστηκαν μέσα στη δεκαετία του 1970 και ήταν κρατούσες μέχρι και το τέλος της δεκαετίας του 1980, όταν άρχισαν να κερδίζουν έδαφος διαφορετικές αντιλήψεις σχετικά με την ανάπτυξη λογισμικού. Στη δομημένη ανάλυση και σχεδίαση κάθε εφαρμογή λογισμικού θεωρείται ως μία ενιαία και σχετικά αυτοτελής, μεγάλη και σύνθετη λειτουργική μονάδα λογισμικού, η οποία είναι εξ ολοκλήρου υπεύθυνη για την επικοινωνία με το περιβάλλον της και για την πραγματοποίηση των υπολογιστικών καθηκόντων της. Αυτά τα υπολογιστικά καθήκοντα αναλύονται σε μια ιεραρχία απλούστερων μονάδων λογισμικού, οι οποίες επιδρούν σε δεδομένα. Τα δεδομένα είναι εντελώς ανεξάρτητα από τις μονάδες λογισμικού που επιδρούν σε αυτά, στις οποίες και επικεντρώνεται η εστίαση της προσοχής όλων των προσεγγίσεων ανάπτυξης λογισμικού της δομημένης ανάλυσης και σχεδίασης.

Στη δομημένη ανάλυση οι απαιτήσεις από το λογισμικό περιγράφονται με τη βοήθεια δομημένου κειμένου, καθώς και με ένα διάγραμμα (μοντέλο παράστασης λογισμικού) το οποίο ονομάζεται «διάγραμμα ροής δεδομένων». Το διάγραμμα αυτό είναι κεντρικό στοιχείο όλων των προσεγγίσεων της οικογένειας της δομημένης ανάλυσης και σχεδίασης και χρησιμεύει ως βάση για την αποκάλυψη της ιεραρχίας των μονάδων λογισμικού που αναφέραμε. Από το διάγραμμα ροής δεδομένων, ακολουθώντας κάποια βήματα, φτάνουμε στο διάγραμμα δομής προγράμματος. Το επίπεδο λεπτομέρειας του διαγράμματος δομής προγράμματος είναι αντίστοιχο με εκείνο του διαγράμματος ροής δεδομένων από το οποίο προήλθε. Στο διάγραμμα αυτό περιγράφεται η ιεραρχία των μονάδων που αποτελούν την εφαρμογή λογισμικού, δηλαδή το ποιες είναι οι μονάδες αυτές, καθώς και η επικοινωνία μεταξύ τους, δηλαδή οι κλήσεις που κάνει κάθε μονάδα σε άλλες και οι παράμετροι που ανταλλάσσονται κατά τις κλήσεις αυτές.

**Σχήμα 7.1** Προϊόντα της δομημένης ανάλυσης και σχεδίασης.





Προκειμένου να μπορεί να κατασκευαστεί ο πηγαίος κώδικας, πρέπει να γίνει μια κατά το δυνατόν λεπτομερής περιγραφή κάθε μονάδας λογισμικού που περιέχεται στο διάγραμμα δομής προγράμματος. Η περιγραφή αυτή γίνεται με τη βοήθεια κάποιας υποθετικής και άτυπης γλώσσας προγραμματισμού, η οποία δεν διαθέτει αυστηρότητα στη σύνταξη και τη γραμματική και αναφέρεται ως ψευδοκώδικας. Τα ενδιάμεσα αυτά προϊόντα έχουν ιδιαίτερη σημασία στη δομημένη ανάλυση και σχεδίαση, αποτελώντας τη ραχοκοκαλιά της σχεδίασης του λογισμικού, όπως φαίνεται στο Σχήμα 7.1.

Όλες οι έννοιες που συγκροτούν το οικοδόμημα που μόλις περιγράψαμε σχετίζονται με τα χαρακτηριστικά των γλωσσών προγραμματισμού που χρησιμοποιούνται στην υλοποίηση της εφαρμογής λογισμικού. Οι μονάδες λογισμικού, σε τελική ανάλυση, δεν είναι παρά οι διαδικασίες (procedures), οι συναρτήσεις (functions) είτε οι υπορουτίνες (subroutines) γλωσσών προγραμματισμού, όπως η Pascal, η C, η FORTRAN και η BASIC.

Με τη σειρά τους, και σε πολύ μεγαλύτερο βαθμό, οι γλώσσες προγραμματισμού σχετίζονται με τα χαρακτηριστικά του υλικού (hardware) των υπολογιστών της εποχής στην οποία αναφερόμαστε: ακολουθιακή εκτέλεση εντολών, απλές δυνατότητες επικοινωνίας με το περιβάλλον μόνο με τη μορφή κειμένου, καθώς και περιορισμένες (σε σχέση με όσα ακολούθησαν) επιδόσεις και αποθηκευτικές ικανότητες. Χαρακτηριστικά όπως η πολυεπεξεργασία, η παράλληλη εκτέλεση εντολών, η επικοινωνία με τον χρήστη με τη βοήθεια περιβάλλοντος γραφικών, η δυνατότητα σύνθεσης και αναπαραγωγής ήχου και εικόνας κ.ά. έγιναν πραγματικότητες μέσα στη δεκαετία του 1990 και, κυρίως, τα λίγα τελευταία χρόνια. Σταδιακά ενσωματώθηκαν στις γλώσσες προγραμματισμού και άσκησαν –με τον τρόπο τους– πίεση για εξέλιξη του τρόπου κατασκευής του λογισμικού, ο οποίος δεν ήταν ούτως ή άλλως χωρίς προβλήματα.

Οι εξελίξεις αυτές έφεραν σε νέο έδαφος την Τεχνολογία Λογισμικού. Η δομημένη ανάλυση και σχεδίαση αποτέλεσε το βάθρο για την εξέλιξη της φιλοσοφίας ανάπτυξης λογισμικού. Σήμερα, η κρατούσα αντίληψη στην ανάπτυξη λογισμικού αναφέρεται ως «αντικειμενοστρεφής τεχνολογία» ή «παράδειγμα» (object-oriented technology, paradigm). Η φιλοσοφία αυτή είναι αρκετά γενική, ώστε η δομημένη ανάλυση και σχεδίαση να μπορεί να

θεωρηθεί υποσύνολό της. Επίσης, μπορεί να υιοθετηθεί από πολλές σύγχρονες μεθοδολογίες και εργαλεία τα οποία υλοποιούν ευέλικτα και προσαρμόσιμα μοντέλα κύκλου ζωής λογισμικού.

Όποιο δρόμο και αν ακολουθήσουμε, το τελικό ζητούμενο είναι να προσδιορίσουμε ποιες είναι εκείνες οι μονάδες λογισμικού που θα «κάνουν τη δουλειά», όπως και αν αυτές ονομάζονται. Ο καθορισμός των προδιαγραφών των απαιτήσεων από το λογισμικό αλλά και οι εργασίες του ελέγχου και της επαλήθευσης του λογισμικού θα εξακολουθήσουν να έχουν την ίδια βαρύνουσα σημασία.

## ΕΝΟΤΗΤΑ 7.2. ΧΑΡΑΚΤΗΡΙΣΤΙΚΑ ΤΟΥ ΣΥΓΧΡΟΝΟΥ ΛΟΓΙΣΜΙΚΟΥ

Πώς είναι το λογισμικό σήμερα; Ποια είναι τα χαρακτηριστικά του; Σε τι θυμίζει τις εφαρμογές που έτρεχαν σε μια δυσνόητη μονόχρωμη οθόνη ενός τερματικού; Ποια είναι τα χαρακτηριστικά των χρηστών του; Αυτά είναι μερικά λογικά ερωτήματα τα οποία αιτιολογούν την παράθεση ορισμένων σύγχρονων χαρακτηριστικών του λογισμικού, προκειμένου να συνθέσουμε μια γενική εικόνα η οποία θα μας είναι χρήσιμη στην αιτιολόγηση ή έστω στην αντίληψη των σύγχρονων εξελίξεων της επιστήμης της Τεχνολογίας Λογισμικού (Σχήμα 7.2).

**Σχήμα 7.2** Κομμάτια που συνθέτουν το παζλ του λογισμικού σήμερα.

Βρίσκεται σε όλες τις συσκευές	Είναι απλό στη χρήση	Λειτουργεί σε περιβάλλον δικτύου
Είναι ανεξάρτητο από το υλικό	Έχει μεγάλη πολυπλοκότητα	Γίνεται απαραίτητο



Πρώτον, σήμερα το λογισμικό βρίσκεται παντού. Αποτελεί μέρος όλων των σύγχρονων ηλεκτρονικών ψηφιακών διατάξεων, ανεξάρτητα από το αν είναι ορατό στον χρήστη αυτών. Μπορεί κανείς να φέρει αναρίθμητα παραδείγματα, από τα ολοένα και μικρότερα ψηφιακά εργαλεία μέχρι το σύστημα ελέγχου της λειτουργίας του κινητήρα του αυτοκινήτου αλλά και του... πλυντηρίου μας.

Δεύτερον, το λογισμικό γίνεται ολοένα και πιο απλό στη χρήση και προσιτό σε οποιονδήποτε. Με τη βοήθεια του λογισμικού οι χρήστες χειρίζονται τις ψηφιακές συσκευές μέσω παραστάσεων που πλησιάζουν αυτές του πραγματικού κόσμου, έχοντας στη διάθεσή τους από παράθυρα, εικόνα και ήχο μέχρι και «εξωτικές» μέχρι πρότινος ικανότητες, όπως αναγνώριση φωνής και εικονική πραγματικότητα.

Τρίτον, σήμερα το λογισμικό στεγάζεται σε δίκτυο, είτε στο αυτοτελές εταιρικό δίκτυο είτε στο καθολικά, πλέον, διαδεδωμένο διαδίκτυο. Σε ειδικές μόνο περιπτώσεις το λογισμικό λειτουργεί αυτοτελώς σε κάποιο μη δικτυωμένο υπολογιστή. Λειτουργώντας σε δικτυακό περιβάλλον, κάθε εφαρμογή λογισμικού επιτελεί τις λειτουργίες για τις οποίες προορίζεται, συνεργαζόμενη και με άλλες εφαρμογές που βρίσκονται κάπου –οπουδήποτε– στο δίκτυο.

Τέταρτον, το λογισμικό τείνει να γίνεται ολοένα και πιο ανεξάρτητο από τη μηχανή στην οποία τρέχει και, μάλιστα, σε επίπεδο εκτελέσιμου κώδικα. Ο κατασκευαστής λογισμικού μπορεί να επικεντρώσει την προσοχή του στη λειτουργικότητα και την ορθότητα των εφαρμογών που κατασκευάζει και όχι στις τεχνικές λεπτομέρειες της μηχανής όπου αυτές θα τρέχουν. Το χαρακτηριστικό αυτό (μεταφερσιμότητα) αποτέλεσε επιδίωξη και σε παλαιότερες εποχές της επιστήμης των υπολογιστών και μερικώς επετεύχθη, σε επίπεδο όμως πηγαίου και όχι εκτελέσιμου κώδικα.

Πέμπτο και λογικά επαγόμενο από τα παραπάνω, το λογισμικό γίνεται ολοένα και πιο πολύπλοκο. Είναι, πλέον, δεδομένη η αύξηση των πεδίων χρήσης του λογισμικού και σε εφαρμογές εκτός εργαστηρίου και επιχειρήσεων. Το λογισμικό έρχεται πιο κοντά στον μη καταρτισμένο χρήστη, γεγονός που δημιουργεί την ανάγκη για ενσωμάτωση εννοιών, συμβόλων

και αναπαραστάσεων που είναι πιο κοντά στην πραγματικότητα που αντιπροσωπεύει το περιβάλλον εφαρμογής του λογισμικού. Δυνατότητες όπως η αναπαράσταση εικονικής πραγματικότητας, η λειτουργία σε ανομοιογενές δικτυακό περιβάλλον, η συνεργασία με άλλες εφαρμογές στο δίκτυο και η ανεξαρτησία από το υλικό δεν είναι απλή υπόθεση να πραγματοποιούνται.

Τέλος, είναι φανερό ότι το λογισμικό γίνεται ολοένα και πιο απαραίτητο και υπεισέρχεται σε κρίσιμους τομείς της ζωής μας. Κρίσιμους όχι μόνο στην επαγγελματική σφαίρα αλλά και στην επικοινωνία, την εκπαίδευση και, τελικά, την ίδια τη λειτουργία του κοινωνικού οικοδομήματος. Χωρίς λογισμικό δεν είναι δυνατόν να λειτουργήσουν τα αεροδρόμια, οι σύγχρονες ιατρικές συσκευές, οι τράπεζες, οι τηλεπικοινωνίες και πολλές άλλες απαραίτητες στο σύγχρονο άνθρωπο υπηρεσίες. Από την άλλη, οφείλουμε να σημειώσουμε ότι, αν ο ιδιωτικός χαρακτήρας των δεδομένων που αφορούν πολλές πλευρές της ζωής μας σήμερα απειλείται, αυτό γίνεται με το λογισμικό ως απαραίτητο εργαλείο στα χέρια όσων την απειλούν. Είναι χρήσιμο, λοιπόν, οι ασχολούμενοι με την ανάπτυξη λογισμικού να συνειδητοποιούμε και τις μη καθαρά τεχνικές πλευρές του αντικειμένου της επιστήμης μας.

## Σύνοψη ενότητας

---

*Το λογισμικό μεταβάλλεται με ραγδαίους ρυθμούς. Τρέχει, πλέον, πάνω σε δίκτυο και γίνεται ολοένα και πιο φιλικό αλλά και απαραίτητο σε πολλές πλευρές της ζωής μας, με όλα τα καλά και τα κακά που αυτό συνεπάγεται. Όλα αυτά επιφέρουν, ασφαλώς, αύξηση της πολυπλοκότητας του λογισμικού, το οποίο, πλέον, λειτουργεί σε ανομοιογενή και κατανεμημένα περιβάλλοντα. Η τεχνολογία λογισμικού βρίσκεται προ μιας νέας πραγματικότητας και καλείται να προσαρμόσει τις προσεγγίσεις της στις νέες αυτές συνθήκες.*

## ΕΝΟΤΗΤΑ 7.3. ΕΞΕΛΙΞΕΙΣ ΚΑΙ ΤΑΣΕΙΣ ΣΤΗΝ ΑΝΑΠΤΥΞΗ ΤΟΥ ΛΟΓΙΣΜΙΚΟΥ

Η κατασκευή λογισμικού με τα χαρακτηριστικά που αναφέρονται στην Ενότητα 1.2 και, ασφαλώς, με τις απαιτήσεις να κάνει τη δουλειά για την οποία προορίζεται και να την κάνει σωστά, να κατασκευάζεται εντός χρονοδιαγράμματος και προϋπολογισμού, να μην περιέχει σφάλματα, να συντηρείται εύκολα αλλά και να είναι επίκαιρο και ανταγωνιστικό ως προϊόν είναι η πρόκληση με την οποία βρίσκονται αντιμέτωποι οι κατασκευαστές λογισμικού σήμερα. Είναι αναμενόμενο ότι πολλά από τα χαρακτηριστικά της διαδικασίας ανάπτυξης λογισμικού πρέπει να εξελίσσονται συνεχώς, ώστε να επιτυγχάνεται το επιθυμητό αποτέλεσμα. Η Τεχνολογία Λογισμικού οφείλει να παρέχει την απαιτούμενη υποστήριξη και τεκμηρίωση, ώστε η κατασκευή του λογισμικού να είναι μια ιδιαίτερα ευέλικτη και προσαρμόσιμη διαδικασία.

Ορισμένες από τις εξελίξεις που έπαιξαν σημαντικό ρόλο στη θεμελίωση της ανάγκης νέων προσεγγίσεων στην ανάπτυξη του λογισμικού είναι οι ακόλουθες:

- Η προσήλωση των κατασκευαστών σε αυστηρές διαδικασίες και πρότυπα που είτε ήταν αποτέλεσμα προσπάθειας σύγκλισης υπάρχουσών πρακτικών είτε είχαν μικρή σχέση με τους κατασκευαστές λογισμικού αποδείχθηκε αναποτελεσματική. Στην πράξη, οι παραδοσιακές προσεγγίσεις ξεπεράστηκαν ως απλά ανεπαρκείς, μια και η ανάπτυξη του λογισμικού σπάνια γίνονταν «κατά γράμμα». Από την άλλη πλευρά, η ανεπάρκεια που μόλις αναφέρθηκε συχνά γεννούσε το αντίθετο φαινόμενο, δηλαδή την πλήρη «αναρχία» στην ανάπτυξη λογισμικού. Και πάλι όμως, η άναρχη ανάπτυξη οδηγούσε (και οδηγεί) με προδιαγεγραμμένη ακρίβεια σε αποτυχημένα έργα ανάπτυξης λογισμικού.
- Ο ρυθμός με τον οποίο αναπτύχθηκε η τεχνολογία του υλικού των υπολογιστών ξεπέρασε κάθε πρόβλεψη, δίνοντας στους χρήστες εργαλεία και δυνατότητες αξιοποίησης σε νέες περιοχές εφαρμογών.

- Η ανάδειξη του διαδικτύου (το οποίο αρχικά δημιουργήθηκε για στρατιωτικούς σκοπούς στις ΗΠΑ) στο νέο μέσο επικοινωνίας του 21ου αιώνα δημιούργησε μια νέα πλατφόρμα πάνω στην οποία τρέχει το λογισμικό. Τα σημαντικότερα χαρακτηριστικά αυτής είναι η ανομοιογένεια, η μεγάλη γεωγραφική κατανομή, η μη ασφαλής επικοινωνία αλλά και η εκρηκτικά αυξανόμενη ζήτηση για υπηρεσίες και εφαρμογές λογισμικού που τρέχουν πάνω σε αυτή τη νέα πλατφόρμα.
- Η σύγκλιση, μετά από αρκετό καιρό, των επικρατέστερων από τις νέες αντικειμενοστρεφείς προσεγγίσεις ανάπτυξης λογισμικού και η επαφή τους με τη βιομηχανική πρακτική.

Μπορούμε, λοιπόν, να αναγνωρίσουμε ότι σε αυτό το πλαίσιο εντάσσονται οι παρακάτω τάσεις στην ανάπτυξη του λογισμικού, οι οποίες έχουν, πλέον, διαφανεί με αρκετή σαφήνεια:

- Τα μοντέλα κύκλου ζωής που ακολουθούνται έχουν γίνει ευέλικτα και παραμετρικά, δηλαδή προσαρμόζονται στα εκάστοτε χαρακτηριστικά του περιβάλλοντος ανάπτυξης λογισμικού αλλά και του θεματικού πεδίου της υπό ανάπτυξη εφαρμογής (application domain).
- Η τεκμηρίωση του λογισμικού, δηλαδή το σύνολο των προϊόντων που περιγράφουν το λογισμικό και παράγονται σε όλες τις φάσεις του κύκλου ζωής αυτού, δεν ακολουθεί αυστηρά πρότυπα δομής όπως στο παρελθόν, αλλά προσαρμόζεται στις εκάστοτε συνθήκες ανάπτυξης, τα εργαλεία που χρησιμοποιούνται και την υπάρχουσα εμπειρία.
- Τα περιβάλλοντα ανάπτυξης (γλώσσες προγραμματισμού, βιβλιοθήκες συστατικών λογισμικού, εργαλεία συγγραφής κώδικα, εργαλεία ελέγχου προγράμματος, βιβλιοθήκες αναφοράς) έχουν γίνει ιδιαίτερα σύνθετα και ολοκληρωμένα, υποστηρίζοντας τον προγραμματιστή σε περισσότερες εργασίες απ' ό,τι στο παρελθόν.
- Ο ρόλος των εργαλείων CASE έχει εξελιχθεί. Τα εργαλεία CASE, πλέον, δεν φιλοδοξούν να αυτοματοποιήσουν πλήρως την παραγωγή λογισμικού. Αρκετά από αυτά συνεργάζονται στενά με τα περιβάλλοντα ανάπτυξης λογισμικού, με σκοπό να εκτελέσουν

σωστά επιρρεπείς σε σφάλματα εργασίες, να υποστηρίξουν την τεκμηρίωση του λογισμικού από τα πρώτα στάδια ανάπτυξης (προδιαγραφή) και την διατήρησή της σε επίκαιρη κατάσταση, να αυτοματοποιήσουν ένα μέρος της παραγωγής πηγαίου κώδικα και, γενικότερα, να υποστηρίξουν τους κατασκευαστές λογισμικού.

Όσα μόλις αναφέρθηκαν δεν αποτελούν, ασφαλώς, κοσμογονία αλλά φυσική εξέλιξη ενός νέου κλάδου της μηχανικής, της Τεχνολογίας Λογισμικού, η οποία εντάσσεται στην οικογένεια τεχνικών επιστημών με τη μεγαλύτερη και πιο ραγδαία ανάπτυξη που γνώρισε μέχρι σήμερα η ιστορία στις επιστήμες της Πληροφορικής. Το ζητούμενο από την Τεχνολογία Λογισμικού παραμένει το ίδιο και, όπως χαρακτηριστικά αναφέραμε στον πρώτο τόμο, δεν είναι άλλο από την εύρεση και θεμελίωση μεθόδων για να περιγράφεται, να κατασκευάζεται και να συντηρείται λογισμικό.

### **Δραστηριότητα I/Κεφάλαιο 7**

Αναφέρατε περιληπτικά, σε λιγότερο από δύο γραμμές για καθεμία, τις τέσσερις εξελίξεις και τις τέσσερις σύγχρονες τάσεις στην ανάπτυξη του λογισμικού. Μπορείτε να αναγνωρίσετε συσχετίσεις μεταξύ τους;



## ΕΝΟΤΗΤΑ 7.4. ΑΔΥΝΑΜΙΕΣ ΤΗΣ ΔΟΜΗΜΕΝΗΣ ΑΝΑΛΥΣΗΣ ΚΑΙ ΣΧΕΔΙΑΣΗΣ

Ποιο από τα χαρακτηριστικά των εφαρμογών λογισμικού καθιστά ανεπαρκή την προσέγγισή τους με τη δομημένη ανάλυση και σχεδίαση; Η απάντηση στο ερώτημα αυτό αποτελεί κλειδί για την αναζήτηση και την κατανόηση μιας νέας φιλοσοφίας προσέγγισης του λογισμικού. Μπορεί κανείς να ανατρέξει εκτενώς στη βιβλιογραφία, προκειμένου να βρει διάφορες απαντήσεις στο ερώτημα. Εμείς θα επιλέξουμε τη μονολεκτική απάντηση «η πολυπλοκότητα». Η πολυπλοκότητα αφορά όχι μόνο την καθαρά υπολογιστική εργασία του λογισμικού αλλά και την ανάγκη διαχείρισης μεγάλου όγκου δεδομένων. Μια εκτενής συζήτηση σχετικά με το θέμα περιέχεται στο βιβλίο του Booch *Object Oriented Analysis and Design with Applications*, το οποίο περιέχεται στη βιβλιογραφία.

Σημεία στα οποία η πολυπλοκότητα του λογισμικού κάνει φανερές τις αδυναμίες της δομημένης ανάλυσης και σχεδίασης μπορούν να αναζητηθούν σε δύο κατευθύνσεις: σε αυτή που αφορά το θεωρητικό και σε αυτή που αφορά το πρακτικό μέρος αυτής.

Στο θεωρητικό επίπεδο, η δομημένη ανάλυση και σχεδίαση παρουσιάζει μια εγγενή αδυναμία στην απεικόνιση των οντοτήτων του πραγματικού κόσμου σε συστατικά λογισμικού. Επιχειρεί να αποσυνθέσει ένα πρόβλημα σε μια ιεραρχία συστατικών λογισμικού, καθένα εκ των οποίων επιτελεί ένα μικρό μέρος της λύσης του. Αυτή η εφαρμογή της αρχαίας ρήσης «διαίρει και βασίλευε» έχει νόημα όταν η λύση του προβλήματος είναι μόνο υπόθεση αριθμητικών υπολογισμών. Στον πραγματικό κόσμο όμως, και ιδιαίτερα στο πεδίο εφαρμογών λογισμικού που σχετίζονται με την επιχειρηματική δραστηριότητα, αυτή η προσέγγιση υστερεί για δύο λόγους:

- Πρώτον, δεν λαμβάνει υπόψη τα δεδομένα, τα οποία έχουν τη δική τους πολύπλοκη δομή και εξαρτήσεις. Τα δεδομένα στη δομημένη ανάλυση και σχεδίαση είναι ανεξάρτητα από τις λειτουργικές μονάδες που επιδρούν σε αυτά, πράγμα που δεν ισχύει στο επίπεδο του πραγματικού κόσμου, όπου η διαχείριση δεδομένων δεν είναι ανεξάρτητη από αυτά.

- Δεύτερον, το υπολογιστικό μοντέλο που αποτελείται από το δίδυμο «συστατικά λογισμικού» και «ανεξάρτητα δεδομένα» δεν αντιστοιχεί σε οντότητες του πραγματικού κόσμου, δηλαδή δεν παριστά οντότητες που είναι αντιληπτές στον πραγματικό κόσμο. Με μια άλλη διατύπωση, δεν μοντελοποιεί εύκολα και φυσικά την επιχειρησιακή λογική (business logic). Ως εκ τούτου, είναι ανεπαρκές ως μεθοδολογικό εργαλείο για τη δόμηση λογισμικού.

Σε πρακτικό επίπεδο μπορούμε να εντοπίσουμε τα ακόλουθα προβλήματα:

- Ο προσδιορισμός των απαιτήσεων είναι το δυσκολότερο από τα βήματα της ανάπτυξης λογισμικού. Η ακριβής περιγραφή των πολλών, πολύπλοκων και αλληλοσυσχετιζόμενων απαιτήσεων με χρήση μετασχηματισμών και ανεξάρτητων δεδομένων, καθώς και η αντιμετώπιση των μεταβολών ακόμα και κατά τη διάρκεια της ανάπτυξης του λογισμικού φέρουν τη δομημένη προσέγγιση στα όριά της.
- Η διαχείριση των μοντέλων παράστασης λογισμικού είναι μια δύσκολη και επιρρεπής σε σφάλματα εργασία. Το πλήθος, η πολυπλοκότητα και οι συσχετίσεις των συστατικών αυτών, η μεταβολή τους με το χρόνο αλλά και οι παρενέργειες κατά την πραγματοποίηση μεταβολών καθιστούν την εργασία αυτή ακόμα δυσκολότερη.
- Η συντήρηση του λογισμικού έχει εξελιχθεί σε μια πολύ δύσκολη διαδικασία που δυσκολεύει περισσότερο καθώς αυξάνεται το μέγεθος του λογισμικού. Έχει αναφερθεί ότι η συνεισφορά της συντήρησης στο συνολικό κόστος κατά τον κύκλο ζωής λογισμικού μπορεί να ξεπεράσει το 50%.
- Μολονότι κατά την ανάπτυξη μιας νέας εφαρμογής ενδέχεται να αναγνωριστούν ομοιότητες με τα χαρακτηριστικά μιας υπάρχουσας, η επαναχρησιμοποίηση συστατικών λογισμικού που έχουν κατασκευαστεί με τη δομημένη ανάλυση και σχεδίαση δεν ενθαρρύνεται. Αυτό ισχύει στη γενική περίπτωση, διότι η πολυπλοκότητα και η φύση των συσχετίσεων ενός συστατικού λογισμικού – μέρους μιας εφαρμογής που κατασκευάστηκε με τη δομημένη φιλοσοφία δεν επιτρέπουν τη γενίκευση και εύκολη επαναχρησιμοποίησή του. Ο κανόνας αυτός ισχύει ιδιαίτερα για λογισμικό που χρησιμοποιείται σε επιχειρηματικές εφαρμογές και, γενικά, σε

εφαρμογές που σχετίζονται με τη διαχείριση δεδομένων, ενώ εξαίρεση αποτελούν οι βιβλιοθήκες μαθηματικών συναρτήσεων που αφορούν καθαρά υπολογιστικές εργασίες.

Όσα αναφέρθηκαν, με κανένα τρόπο δεν σημαίνουν δύο πράγματα: πρώτον, ότι η δομημένη ανάλυση και σχεδίαση είναι άχρηστη, εσφαλμένη ή κάτι τέτοιο και, δεύτερον, ότι η αντικειμενοστρεφής φιλοσοφία έχει τη λύση σε όλα τα προβλήματα της ανάπτυξης του λογισμικού. Όπως εξάλλου αναφέρθηκε, η συζήτηση αυτή έχει το νόημα της αντίληψης των εξελίξεων που εισήγαγαν μια νέα φιλοσοφία προσέγγισης του λογισμικού. Πολλά από τα προβλήματα που υπήρχαν στην ανάπτυξη του λογισμικού εξακολουθούν να υπάρχουν με τη μία ή με την άλλη έννοια, ενδεχομένως σε μικρότερο βαθμό. Η αντικειμενοστρεφής τεχνολογία είναι μόνο ένα καλύτερο εργαλείο στα χέρια του ανθρώπου για τη λύση των προβλημάτων, δεν είναι ίδια η λύση των προβλημάτων.

## **Δραστηριότητα 2/Κεφάλαιο 7**

Περιγράψτε, χρησιμοποιώντας λιγότερες από 100 λέξεις συνολικά, τα έξι σημεία που καθιστούν σήμερα δύσκολη την ανάπτυξη λογισμικού με τη δομημένη ανάλυση και σχεδίαση.



## ΕΝΟΤΗΤΑ 7.5. ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ ΤΗΣ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΟΥΣ ΤΕΧΝΟΛΟΓΙΑΣ

Αξίζει να σημειώσουμε από την αρχή ότι η ιδέα της αντικειμενοστρεφούς προσέγγισης υπήρχε από τη δεκαετία του '60 και η πρώτη γλώσσα που είχε τέτοια χαρακτηριστικά ήταν η Simula-67. Τότε κανείς σχεδόν δεν έδινε μεγάλη σημασία στις ιδέες που εισήγαγε η Simula: η κοινότητα του λογισμικού είχε στραμμένη την προσοχή της στις ιδέες της δομημένης ανάλυσης/σχεδίασης. Άλλες γλώσσες προγραμματισμού που είχαν τέτοια στοιχεία ήταν η Alphard, η CLU, η ADA (η οποία γνώρισε μεγάλη διάδοση, διότι υποστηρίχτηκε από το υπουργείο άμυνας των ΗΠΑ) και η Smalltalk.

Η προσέγγιση της αντικειμενοστρεφούς τεχνολογίας μπορεί να γίνει από διάφορους εναλλακτικούς δρόμους. Την εποχή που παρατηρήθηκε έκρηξη ενδιαφέροντος για το θέμα δημοσιεύτηκαν πολλές θεωρητικές αναλύσεις με παρεμφερή ορολογία και με λιγότερο ή περισσότερο συγκεκριμένες αναφορές στις πρακτικές πλευρές της αντικειμενοστρεφούς προσέγγισης στον προγραμματισμό. Ο αναγνώστης παραπέμπεται στη βιβλιογραφία για εκτενέστερες αναφορές στο θέμα.

### 7.5.1. Ορισμοί

Μέχρι το σημείο αυτό χρησιμοποιήσαμε τον όρο «αντικειμενοστρεφής» ως μετάφραση του αγγλικού όρου «object-oriented». Καιρός είναι να δώσουμε έναν πιο σαφή ορισμό και να εισάγουμε την ορολογία που σχετίζεται με την αντικειμενοστρεφή τεχνολογία. Θα προσεγγίσουμε το θέμα ξεκινώντας από την κορυφή της πυραμίδας.

#### **Αντικειμενοστρεφής:**

Αντικειμενοστρεφής (ο, η), αντικειμενοστρεφές (το) είναι χαρακτηρισμός που σημαίνει «στραμμένος (προσανατολισμένος) σε αντικείμενα» και αποδίδεται σε εκείνο τον τρόπο σκέψης κατά την ανάπτυξη λογισμικού στον οποίο τα «αντικείμενα» είναι οι βασικές δομικές μονάδες του λογισμικού.

Θα σημειώσουμε ότι επιλέξαμε να χρησιμοποιήσουμε το επίθεμα «-στρεφής» και όχι «-στραφής» όπως αναφέρεται σε άλλα σημεία στη βιβλιογραφία, σύμφωνα με τη λογική που αυτό χρησιμοποιείται στο δόκιμο «εσωστρεφής» (και όχι «εσωστραφής»), που σημαίνει «στροφή προς τον εαυτό». Μια άλλη, κατά τη γνώμη μας άστοχη, απόδοση του όρου στην ελληνική γλώσσα είναι «αντικειμενικός», λέξη που υπήρχε ήδη στην ελληνική γλώσσα και σημαίνει κάτι εντελώς διαφορετικό.

Ο παραπάνω ορισμός χρησιμοποίησε τον όρο «αντικείμενο», ο οποίος θα πρέπει επίσης να οριστεί στο πλαίσιο του λογισμικού.

### **Αντικείμενο:**

Ένα αντικείμενο είναι ένα δομικό συστατικό του λογισμικού. Κάθε αντικείμενο έχει κατάσταση, συμπεριφορά και ταυτότητα. Η κατάσταση περιγράφει όλες τις στατικές ιδιότητες του αντικειμένου, όπως τιμές σε μεταβλητές μνήμης. Οι ιδιότητες αυτές λαμβάνουν τιμές ως αποτέλεσμα της συμπεριφοράς του αντικειμένου, δηλαδή του τρόπου με τον οποίο αυτό ανταποκρίνεται σε κλήσεις από το περιβάλλον του, δηλαδή τα άλλα αντικείμενα στο ίδιο περιβάλλον εκτέλεσης. Η ταυτότητα είναι η μοναδική διάκριση του αντικειμένου από τα ομοειδή του.

Από τεχνικής πλευράς, το αντικείμενο μπορεί να θεωρηθεί ως η συγχώνευση δύο εννοιών που κατέχουν δεσπόζουσα θέση στη δομημένη ανάλυση, σχεδίαση και προγραμματισμό: της εγγραφής (struct στη C) και της συνάρτησης ή διαδικασίας (function, procedure στην Pascal). Η εγγραφή είναι μια συλλογή μεταβλητών μνήμης, οι οποίες αποτελούν τη δομή της και κάθε στιγμή έχουν κάποια συγκεκριμένη τιμή. Το σύνολο των τιμών των μεταβλητών μιας εγγραφής αποτελεί την κατάσταση αυτής. Αν στην εγγραφή, όπως την ορίσαμε μέχρι τώρα, προσθέσουμε ενεργά συστατικά λογισμικού, δηλαδή μονάδες προγράμματος που εκτελούν υπολογισμούς, τότε μιλάμε για αντικείμενο και όχι για εγγραφή. Ένα αντικείμενο, δηλαδή, περικλείει ένα σύνολο δεδομένων και ένα σύνολο συναρτήσεων που χειρίζονται τα δεδομένα αυτά και επιτελούν τις λειτουργίες του πεδίου ευθύνης αυτού.

### **Κλάση:**

Μια αφηρημένη περιγραφή της δομής και συμπεριφοράς μιας έννοιας του πραγματικού κόσμου η οποία απεικονίζεται στο λογισμικό ονομάζεται «κλάση». Το σύνολο των αντικειμένων μιας κλάσης έχουν την ίδια δομή και συμπεριφορά.

Θα πρέπει να διακρίνουμε την έννοια «δομή» και την έννοια «κατάσταση». Η δομή χαρακτηρίζεται από το ποιες και τι τύπου μεταβλητές περιγράφουν τις ιδιότητες του αντικειμένου, ενώ η κατάσταση είναι ένα σύνολο συγκεκριμένων τιμών στις μεταβλητές αυτές. Η διατύπωση «ίδια συμπεριφορά» σημαίνει «ίδια απόκριση στο ίδιο εξωτερικό ερέθισμα». Η κλάση είναι μια αφηρημένη έννοια, όπως αφηρημένη είναι η έννοια του «τύπου» (type) στις γλώσσες προγραμματισμού. Ο τύπος δεν είναι κάτι που υπάρχει την ώρα της εκτέλεσης ενός προγράμματος. Αυτό που υπάρχει είναι οι μεταβλητές μνήμης, καθεμία εκ των οποίων λέμε ότι «είναι» κάποιου τύπου.

Μπορούμε, λοιπόν, να αναγνωρίσουμε αντιστοιχία μεταξύ των ζευγαριών τύπος – μεταβλητή και κλάση – αντικείμενο. Όπως ισχύει ότι «κάθε μεταβλητή μνήμης είναι κάποιου τύπου», έτσι ισχύει ότι «κάθε αντικείμενο ανήκει σε μία κλάση», η οποία καθορίζει πλήρως τη δομή και τη συμπεριφορά του. Με βάση τα παραπάνω, μια κλάση μπορεί να θεωρηθεί ως η περιγραφή της δομής και της συμπεριφοράς των αντικειμένων που ανήκουν σε αυτή, τα οποία είναι τα συστατικά στοιχεία λογισμικού που έχουν τη δομή και εκδηλώνουν τη συμπεριφορά.

### **Στιγμιότυπο, εκδοχή:**

Κάθε αντικείμενο αποτελεί ένα μοναδικό και συγκεκριμένο στιγμιότυπο ή εκδοχή (instance) της κλάσης στην οποία ανήκει.

Η κατάσταση ενός αντικειμένου καθορίζεται από τις μεταβλητές κατάστασης οι οποίες αντιστοιχούν στα ιδιώματά του.

### **Πεδίο:**

Ένα πεδίο (field ή attribute) είναι μια μεταβλητή η οποία παριστάνει ένα ιδίωμα του αντικειμένου. Το σύνολο των τιμών όλων των πεδίων ενός αντικειμένου αποτελεί την κατάσταση αυτού.

Η συμπεριφορά ενός αντικειμένου καθορίζεται από τον τρόπο με τον οποίο αυτό αντιδρά σε εξωτερικά ερεθίσματα, δηλαδή σε κλήσεις από τα αντικείμενα που αποτελούν το περιβάλλον του.

### **Μέθοδος:**

Μια μέθοδος είναι ένα ενεργό συστατικό λογισμικού (συνάρτηση) το οποίο υλοποιεί ένα στοιχείο συμπεριφοράς των αντικειμένων μιας κλάσης. Το σύνολο όλων των μεθόδων μιας κλάσης καθορίζει τη συμπεριφορά των αντικειμένων αυτής μέσα σε ένα περιβάλλον εκτέλεσης λογισμικού.

Μια χαρακτηριστική ιδιότητα των αντικειμένων, βασικό στοιχείο της αντικειμενοστρεφούς φιλοσοφίας, είναι η κελυφοποίηση (encapsulation) ή, όπως ισοδύναμα αναφέρεται στη βιβλιογραφία, η απόκρυψη πληροφοριών (information hiding). Στην ελληνική βιβλιογραφία η έννοια απαντάται και με τον όρο «ενθυλάκωση».

### **Κελυφοποίηση, απόκρυψη πληροφοριών:**

Η απόκρυψη των λεπτομερειών υλοποίησης ενός αντικειμένου από το περιβάλλον του μέσω της ελεγχόμενης ορατότητας των πεδίων και των μεθόδων αυτού από άλλα αντικείμενα.

Αρκετοί συγγραφείς, όπως ο Booch, συνδέουν την κελυφοποίηση με την πρόσβαση σε ορισμένα μόνο πεδία και μεθόδους ενός αντικειμένου τα οποία ορίζουν τη διεπαφή (interface) αυτού με τον έξω κόσμο. Αυτή είναι η πρακτική εφαρμογή της απόκρυψης πληροφοριών, η οποία αποτελεί

μία βασική αρχή της αντικειμενοστρεφούς φιλοσοφίας που εφαρμόζεται με διάφορους τρόπους στις γλώσσες προγραμματισμού οι οποίες την υποστηρίζουν.

Ας σημειωθεί ότι επιλέξαμε να χρησιμοποιήσουμε τον όρο «αντικείμενο» και όχι «κλάση», διότι η κλάση είναι μια γενική έννοια, η οποία δεν έχει δικές της λεπτομέρειες υλοποίησης αλλά περιέχει την περιγραφή των λεπτομερειών αυτών ώστε να δημιουργούνται αντικείμενα, δηλαδή συγκεκριμένα στιγμιότυπα της κλάσης. Στις αναφορές που θα ακολουθήσουν, θα μιλάμε για «κλάσεις» όταν η συζήτηση μπορεί να διεξάγεται στο γενικό επίπεδο της περιγραφής των αντικειμένων, ενώ όταν είναι αναγκαίο να αναφερθούμε σε συγκεκριμένες εκδοχές, θα χρησιμοποιούμε τον όρο «αντικείμενο».

## Παράδειγμα Ι/Κεφάλαιο 7

Θα επανέλθουμε στη μελέτη περίπτωσης του πρώτου τόμου του βιβλίου, όπου ασχοληθήκαμε με την εφαρμογή λογισμικού «Επίκουρος», η οποία είχε ως αντικείμενο την υποστήριξη των εργασιών της γραμματείας μιας εκπαιδευτικής μονάδας. Θα διακρίνουμε δύο κλάσεις από την εφαρμογή αυτή –προς το παρόν με τρόπο αυθαίρετο–, έχοντας σκοπό όχι να μάθουμε να αναγνωρίζουμε κλάσεις, αλλά να αποκτήσουμε μια πρώτη άποψη.

Πρόκειται για τις κλάσεις «καθηγητής» και «μάθημα». Γνωρίζουμε τα πεδία τα οποία περιέχονται στην εγγραφή κάθε τέτοιας οντότητας. Οι μέθοδοι τις οποίες αποδίδουμε σε καθεμία από τις κλάσεις αυτές δεν είναι παρά οι λειτουργίες που επιτελούνται επί των πεδίων τους που αναφέρονται στις λειτουργικές απαιτήσεις της εν λόγω μελέτης περίπτωσης του πρώτου τόμου.

Οι κλάσεις που ορίσαμε φαίνονται στο Σχήμα 7.3. Ο συμβολισμός που χρησιμοποιείται είναι προφανής: στο πρώτο διαμέρισμα φιλοξενείται το όνομα της κλάσης, στο δεύτερο, τα ονόματα των πεδίων, και στο τρίτο, τα ονόματα των μεθόδων. Προσπερνάμε, προς το παρόν, τις λεπτομέρειες του συμβολισμού.

**Σχήμα 7.3** Δύο κλάσεις από την εφαρμογή λογισμικού «Επίκουρος».

Καθηγητής
-Αρ_Ταυτότητας -Επώνυμο -Όνομα -Διεύθυνση -Τηλέφωνο
+Προσθήκη_Καθηγητή() +Διαγραφή_Καθηγητή() +Μεταβολή_Στοιχείων()

Μάθημα
-Κωδικός_Μαθήματος -Θεματική_Ενότητα -Τίτλος -Διδάσκων
+Προσθήκη_Μαθήματος() +Μεταβολή_Στοιχείων_Μαθήματος() +Ανάθεση_Μαθήματος() +Διαγραφή_Μαθήματος()

Αντικείμενα που ανήκουν στην κλάση «καθηγητής» αντιστοιχούν σε συγκεκριμένες εκδοχές (instances) αυτής, εν προκειμένω σε πραγματικούς καθηγητές. Επίσης, αντικείμενα που ανήκουν στην κλάση «μάθημα» αντιστοιχούν σε συγκεκριμένα μαθήματα. Τέτοια αντικείμενα εικονίζονται στο σχήμα 7.4. Η κατάσταση των αντικειμένων, δηλαδή το χαρακτηριστικό εκείνο που τα προσδιορίζει και τα διαφοροποιεί μεταξύ τους, σημειώνεται στο με έντονους χαρακτήρες. Η συμπεριφορά τους, όπως αναφέρθηκε, είναι κοινή και εξαρτάται μόνο από την κλάση στην οποία ανήκουν.



**Σχήμα 7.4** Αντικείμενα – στιγμιότυπα των κλάσεων που φαίνονται στο Σχήμα 7.3.

<b>Καθηγητής001</b> -Αρ_Ταυτότητας: <b>ABI23456</b> -Επώνυμο: <b>Βασιλείου</b> -Όνομα: <b>Βασίλειος</b> -Διεύθυνση: <b>Αγ.Βασιλείου Ι</b> -Τηλέφωνο: <b>2101234567</b> +Προσθήκη_Καθηγητή() +Διαγραφή_Καθηγητή() +Μεταβολή_Στοιχείων()	<b>Καθηγητής002</b> -Αρ_Ταυτότητας: <b>AB098765</b> -Επώνυμο: <b>Γεωργίου</b> -Όνομα: <b>Γεώργιος</b> -Διεύθυνση: <b>Αγ.Γεωργίου Ι</b> -Τηλέφωνο: <b>210567890</b> +Προσθήκη_Καθηγητή() +Διαγραφή_Καθηγητή() +Μεταβολή_Στοιχείων()
<b>Μάθημα001</b> -Κωδικός_Μαθήματος: <b>MI501</b> -Θεματική_Ενότητα: <b>ΘΕ24</b> -Τίτλος: <b>Τεχνολογία Λογισμικού</b> -Διδάσκων: <b>ABI23456</b> +Προσθήκη_Μαθήματος() +Μεταβολή_Στοιχείων_Μαθήματος() +Ανάθεση_Μαθήματος() +Διαγραφή_Μαθήματος()	<b>Μάθημα002</b> -Κωδικός_Μαθήματος: <b>MI521</b> -Θεματική_Ενότητα: <b>ΘΕ24</b> -Τίτλος: <b>Βάσεις Δεδομένων</b> -Διδάσκων: <b>AB098765</b> +Προσθήκη_Μαθήματος() +Μεταβολή_Στοιχείων_Μαθήματος() +Ανάθεση_Μαθήματος() +Διαγραφή_Μαθήματος()

### Δραστηριότητα 3/Κεφάλαιο 7

Σε αντιστοιχία με το Παράδειγμα Ι, προσπαθήστε να ορίσετε την κλάση «σπουδαστής», χρησιμοποιώντας όσα αναφέρονται στις Ενότητες 4.3, 4.4.3 και 4.4.5 του Κεφαλαίου 4. Επίσης, δώστε ένα παράδειγμα ενός αντικειμένου της κλάσης αυτής.

### Άσκηση Ι/Κεφάλαιο 7

Χαρακτηρίστε ως «Σωστό» ή «Λάθος» καθεμία από τις παρακάτω προτάσεις.

	Σωστό	Λάθος
1. Μια κλάση περιέχει αντικείμενα.		
2. Μια κλάση περιέχει μεθόδους και πεδία.		
3. Μια κλάση είναι το στιγμιότυπο των αντικειμένων της.		
4. Κάθε μέθοδος ορίζεται στον ορισμό του αντικειμένου όπου ανήκει.		
5. Οι μέθοδοι αντιστοιχούν στην κατάσταση μιας κλάσης.		
6. Οι τιμές των πεδίων περιγράφουν την κατάσταση ενός αντικειμένου.		
7. Η συμπεριφορά όλων των αντικειμένων μιας κλάσης είναι ίδια.		
8. Η μέθοδος είναι μια συνάρτηση ή διαδικασία ενσωματωμένη μέσα σε μια κλάση.		
9. Αντικείμενο = (εγγραφή) + (συναρτήσεις/ διαδικασίες).		
10. Ένα αντικείμενο χαρακτηρίζεται από κατάσταση και συμπεριφορά.		
11. Η έννοια της κλάσης είναι αντίστοιχη της έννοιας της μεταβλητής μνήμης.		
12. Η απόκρυψη πληροφοριών σχετίζεται με την τοποθέτηση μεθόδων και πεδίων μέσα στο κέλυφος μιας κλάσης		

### 7.5.2. Σχέσεις μεταξύ κλάσεων

Αφού έχουμε αναγνωρίσει από τη μέχρι τώρα ενασχόλησή μας με την Τεχνολογία Λογισμικού και τον προγραμματισμό σχέσεις μεταξύ δεδομένων και ενεργών συστατικών λογισμικού, είναι φυσικό να περιμένουμε ότι και μεταξύ των κλάσεων θα υπάρχουν σχέσεις. Όταν μεταξύ δύο κλάσεων αναγνωρίζουμε μια σχέση, τότε η σχέση αυτή υλοποιείται στο επίπεδο των αντικειμένων των κλάσεων αυτών. Μπορούμε να προσπαθήσουμε να μαντέψουμε το είδος των σχέσεων που υπάρχουν μεταξύ κλάσεων κάνοντας τις ακόλουθες δύο παρατηρήσεις.

- Πρώτον, επειδή και οι κλάσεις, όπως οι εγγραφές (records) και οι πίνακες (tables), περιέχουν δεδομένα, θα πρέπει με κάποιο τρόπο να παριστάνονται οι σχέσεις μεταξύ των δεδομένων.
- Δεύτερον, εισάγαμε την έννοια της κλάσης αναφερόμενοι στην ανεπάρκεια της δομημένης φιλοσοφίας να μοντελοποιήσει τον πραγματικό κόσμο. Φυσικό είναι, λοιπόν, να περιμένουμε ότι οι κλάσεις προσφέρουν δυνατότητες καλύτερης μοντελοποίησης του κόσμου μέσα από την εσωτερική δομή αλλά και από τις δυνατές συσχετίσεις μεταξύ τους.

Πράγματι, οι σχέσεις μεταξύ των κλάσεων επαληθεύουν τις δύο αυτές παρατηρήσεις. Τρεις είναι οι κύριοι τύποι σχέσεων μεταξύ κλάσεων: η συσχέτιση (association), η κληρονομικότητα (inheritance) και η συναρμολόγηση (aggregation). Θα παρουσιάσουμε τις σχέσεις αυτές χωρίς να σκοπεύουμε να δείξουμε πώς τις αναγνωρίζουμε, θέμα στο οποίο θα αναφερθούμε σε επόμενο κεφάλαιο.

#### **Συσχέτιση**

Η συσχέτιση (association) είναι η πιο γενική από τις σχέσεις που μπορεί να συνδέει δύο κλάσεις. Μπορούμε με αρκετή ασφάλεια να αντιληφθούμε τη συσχέτιση ως μια πιο γενική απ' ό,τι στο σχεσιακό μοντέλο σχέση μεταξύ δεδομένων. Λέμε «πιο γενική», διότι η συσχέτιση δεν περιορίζεται σε τρεις δυνατούς τύπους, όπως στο σχεσιακό μοντέλο, αλλά έχει οποιαδήποτε χαρακτηριστικά απαιτεί η φύση του προβλήματος.

Μια συσχέτιση χαρακτηρίζεται από:

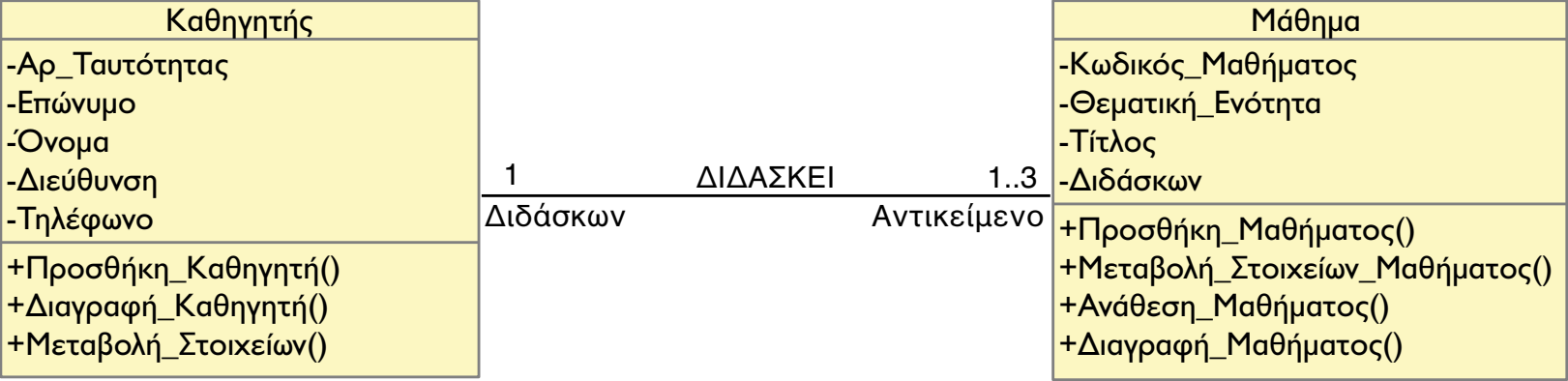
- ένα όνομα, ενδεχομένως διαφορετικό ανάλογα με τη φορά ανάγνωσης,
- την πολλαπλότητα, δηλαδή τον επιτρεπόμενο αριθμό των μελών κάθε μέρους της συσχέτισης και
- τα ονόματα του ρόλου (role) που παίζει κάθε μέρος στη συσχέτιση.

Οι πολλαπλότητες «ένα προς ένα», «ένα προς πολλά» και «πολλά προς πολλά» είναι και στην περίπτωση αυτή οι πιο γενικές, ωστόσο, όπως θα δούμε και στο παράδειγμα που θα ακολουθήσει, μπορούμε να ορίσουμε οποιαδήποτε πολλαπλότητα απαιτείται στις εκάστοτε συνθήκες.

### **Παράδειγμα 2/Κεφάλαιο 7**

Στο Σχήμα 7.5 φαίνεται η συσχέτιση των κλάσεων «καθηγητής» και «μάθημα». Ο ρόλος του «καθηγητή» είναι «διδάσκων», ενώ το «μάθημα» είναι το αντικείμενο της διδασκαλίας. Κάθε καθηγητής μπορεί να διδάσκει από ένα έως τρία το πολύ μαθήματα. Η πολλαπλότητα αυτή της συσχέτισης φαίνεται επίσης στο σχήμα.

Σχήμα 7.5 Μια συσχέτιση μεταξύ κλάσεων.



## Κληρονομικότητα ή γενίκευση

Δύο κλάσεις συνδέονται με κληρονομικότητα όταν η μία αποδίδει στην άλλη τα χαρακτηριστικά της, δηλαδή, κατά κάποιο τρόπο, της τα κληροδοτεί. Η πρώτη κλάση λέγεται κλάση-γονέας, ενώ η δεύτερη, κλάση-παιδί, και μπορεί να προσθέτει στα χαρακτηριστικά που κληρονομεί (πεδία και μεθόδους) και δικά της. Μια κλάση-γονέας μπορεί να αποδίδει τα χαρακτηριστικά της σε πολλές κλάσεις-παιδιά. Αλλού στη βιβλιογραφία θα δείτε ισοδύναμα τον όρο «κλάση-πατέρας», ο οποίος όμως είναι λιγότερο «πολιτικά ορθός».

Στην περίπτωση που κάθε κλάση που συμμετέχει σε σχέση κληρονομικότητας έχει μόνο έναν γονέα, τότε μιλάμε για απλή κληρονομικότητα (single inheritance). Όταν μια κλάση κληρονομεί χαρακτηριστικά από περισσότερες της μίας κλάσεις-γονέα, τότε μιλάμε για πολλαπλή κληρονομικότητα (multiple inheritance). Υπάρχουν πολλές πρακτικές πλευρές στην υλοποίηση της κληρονομικότητας μεταξύ κλάσεων, οι οποίες αφορούν την κάθε συγκεκριμένη γλώσσα προγραμματισμού που υποστηρίζει κληρονομικότητα και δεν θα μας απασχολήσουν εδώ.

Κοιτώντας από πάνω προς τα κάτω, δηλαδή από την κλάση-γονέα στην κλάση-παιδί, μιλάμε για κληρονομικότητα, διότι η κλάση-παιδί έχει ό,τι χαρακτηριστικά έχει η κλάση-γονέας. Όπως αναφέρθηκε, έχει και κάποια επιπλέον χαρακτηριστικά τα οποία δεν έχει ο γονέας (η περίπτωση που μια κλάση κληρονομεί τα χαρακτηριστικά μιας άλλης χωρίς να προσθέτει τίποτε δικό της δεν εξετάζεται ως τετριμμένη). Τα επιπλέον χαρακτηριστικά εξειδικεύουν την κληρονομούμενη κλάση, δηλαδή η κλάση-παιδί είναι μια εξειδίκευση της κλάσης-γονέα.

Κοιτώντας τώρα τη σχέση κληρονομικότητας από κάτω προς τα πάνω, παρατηρούμε ότι η κλάση-γονέας έχει λιγότερα χαρακτηριστικά από την κλάση-παιδί, δηλαδή αποτελεί γενίκευσή της. Με την έννοια αυτή, μιλάμε ισοδύναμα για κληρονομικότητα και γενίκευση. Και στις δύο περιπτώσεις έχουμε στη διάθεσή μας έναν μηχανισμό ταξινόμησης των οντοτήτων που πραγματικού κόσμου που αντιστοιχούν σε κλάσεις από το γενικό στο ειδικό (κληρονομικότητα) ή, ισοδύναμα, από το ειδικό στο γενικό (γενίκευση).

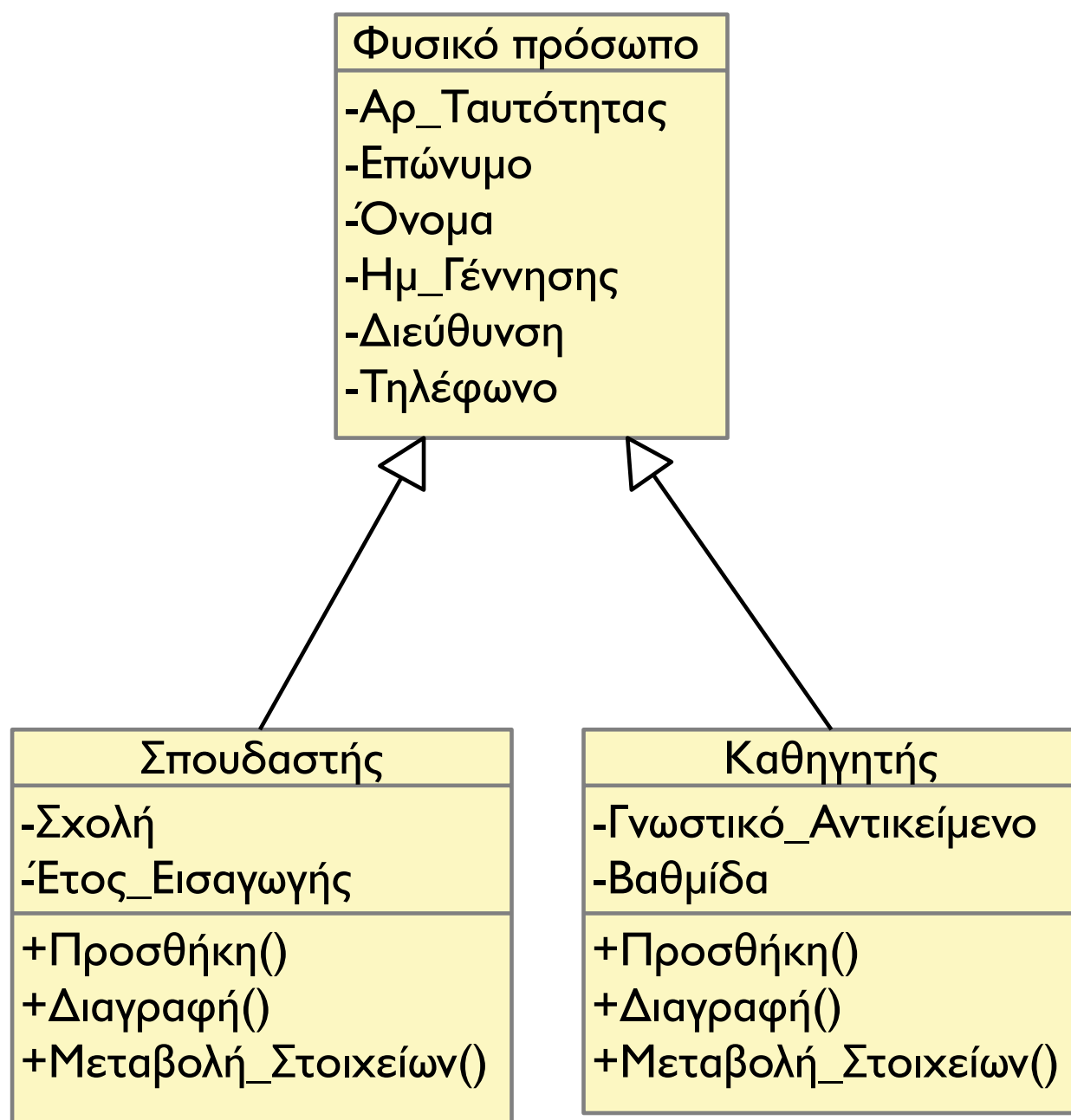
Στην αντικειμενοστρεφή τεχνολογία, κληρονομικότητα και γενίκευση αποτελούν τις δύο όψεις ενός μηχανισμού ταξινόμησης (classification) των εννοιών του πραγματικού κόσμου.

Αυτός ο μηχανισμός ταξινόμησης αποτελεί κεντρικό στοιχείο της αντικειμενοστρεφούς φιλοσοφίας. Το τελευταίο διάστημα γίνεται περισσότερος λόγος για γενίκευση και λιγότερος για κληρονομικότητα (αν και πρόκειται για το ίδιο πράγμα), διότι η γενίκευση μπορεί να γίνει ευκολότερα αντιληπτή ως εργαλείο μοντελοποίησης.

### Παράδειγμα 3/Κεφάλαιο 7

Στο παράδειγμα και στη δραστηριότητα που προηγήθηκαν, ορίσαμε τις οντότητες «καθηγητής» και «σπουδαστής». Αμφότερες περιγράφουν φυσικά πρόσωπα τα οποία έχουν κοινά χαρακτηριστικά (λ.χ. όνομα, επώνυμο). Μπορούμε, λοιπόν, να θεωρήσουμε ότι τα κοινά χαρακτηριστικά που έχουν όλα τα φυσικά πρόσωπα περιγράφονται σε μια κλάση-γονέα «φυσικό πρόσωπο», η οποία τα κληροδοτεί στις κλάσεις «καθηγητής» και «σπουδαστής», όπως φαίνεται στο Σχήμα 7.6.

**Σχήμα 7.6** Ένα παράδειγμα κληρονομικότητας ή γενίκευσης.





Να σημειώσουμε ότι στο Σχήμα 7.6 προσθέσαμε στην κλάση «καθηγητής» τα πεδία «ειδικότητα» και «βαθμίδα». Διαβάζοντας το σχήμα από πάνω προς τα κάτω, διαπιστώνουμε ότι καθεμία από τις έννοιες «καθηγητής» και «σπουδαστής» είναι ειδικότερη από την έννοια «φυσικό πρόσωπο» και κληρονομεί όλα τα χαρακτηριστικά της. Γι' αυτό μιλάμε για κληρονομικότητα. Αντίστροφα, θεωρώντας το σχήμα από κάτω προς τα πάνω, παρατηρούμε ότι η έννοια «φυσικό πρόσωπο» είναι γενικότερη από την έννοια «σπουδαστής» και από την έννοια «καθηγητής», οπότε μιλάμε για γενίκευση.

## **Συναρμολόγηση**

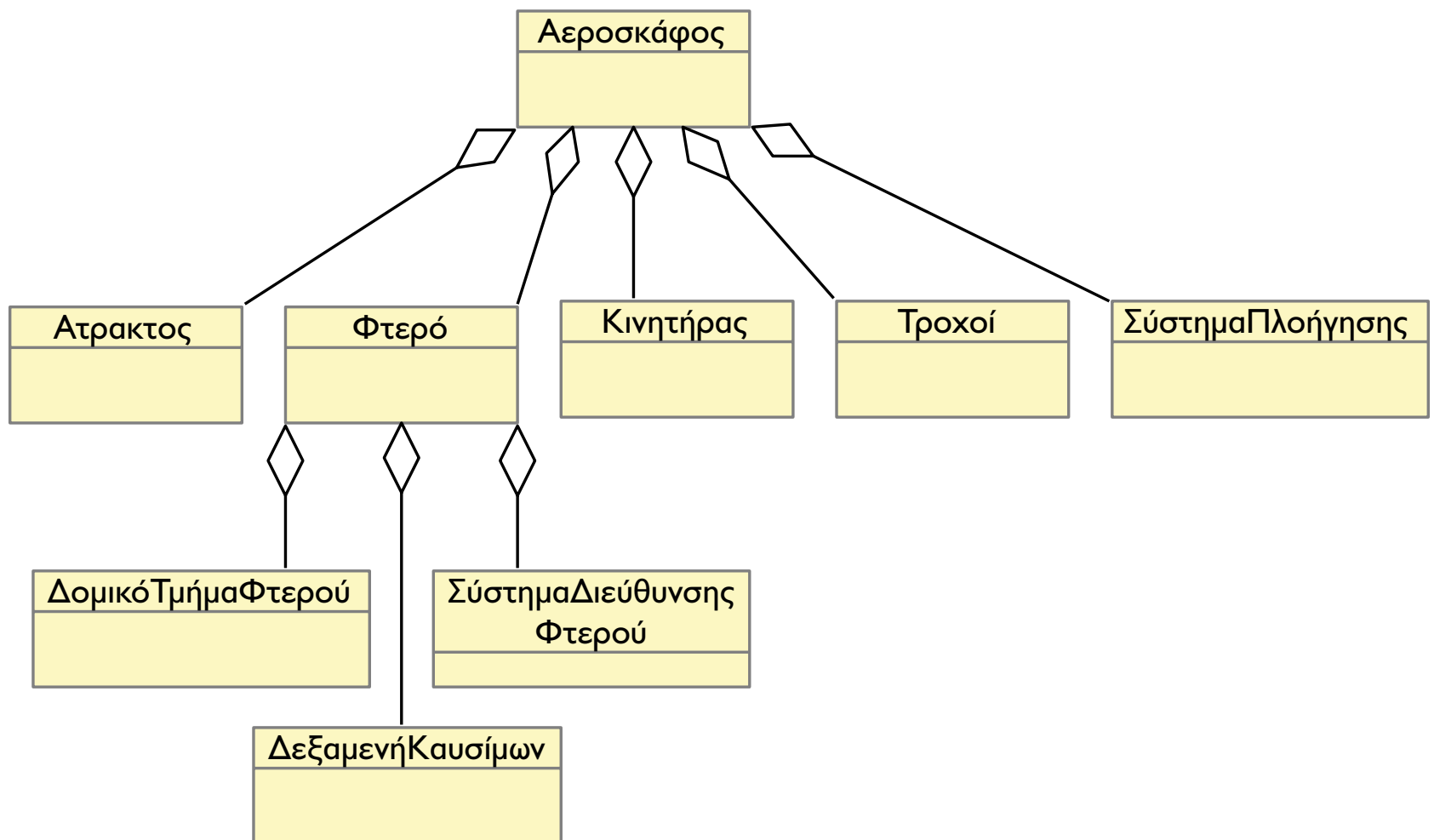
Η συναρμολόγηση ή συνάθροιση (aggregation) είναι εκείνη η σχέση που εκφράζει ακριβώς αυτό που λέει το όνομά της: τη σύνθεση συνόλων από απλούστερα μέρη. Η έννοια της συναρμολόγησης δεν μπορεί να θεωρηθεί ως σχέση κληρονομικότητας, διότι δεν αναγνωρίζουμε χαρακτηριστικά που αποδίδονται από μία κλάση σε μία άλλη. Ενδεχομένως, η συναρμολόγηση να μπορεί να θεωρηθεί ως μια ειδική συσχέτιση την οποία όμως έχει νόημα να εξετάσουμε αυτοτελώς, διότι είναι ένα χρήσιμο εργαλείο για τη μοντελοποίηση του πραγματικού κόσμου.

## Παράδειγμα 4/Κεφάλαιο 7

Θα αναγκαστούμε να παρακάμψουμε το γνωστό μας πρόβλημα της εφαρμογής λογισμικού «Επίκουρος», διότι σε αυτή δεν διακρίνουμε καμία σχέση συναρμολόγησης που θέλουμε να περιγράψουμε εδώ. Όπως μπορούμε να αντιληφθούμε, δεν είναι υποχρεωτικό σε κάθε πρόβλημα να αναγνωρίζονται όλες οι δυνατές σχέσεις.

Ας θεωρήσουμε, λοιπόν, μια εφαρμογή λογισμικού που διαχειρίζεται τη συντήρηση των αεροσκαφών που χρησιμοποιούνται στις αερομεταφορές. Η συντήρηση απαιτεί την παράσταση της δομής κάθε αεροσκάφους, η οποία μπορεί να γίνει σε πάρα πολλά επίπεδα, ξεκινώντας από την περισσότερο σύνθετη οντότητα (ολόκληρο το αεροσκάφος) και φτάνοντας μέχρι και το μικρότερο εξάρτημα. Στο Σχήμα 7.7 φαίνεται μια τέτοια σχέση συναρμολόγησης σε ένα πρώτο, πολύ γενικό επίπεδο.

**Σχήμα 7.7** Ένα πολύ μικρό τμήμα της συναρμολόγησης ενός αεροσκά-  
φους από επιμέρους συστατικά.



Οι αριθμοί πάνω στις άκρες των ευθύγραμμων τμημάτων που παριστάνουν τις σχέσεις αντιστοιχούν στην πολλαπλότητα των αντίστοιχων συστατικών. Κάθε αεροσκάφος αποτελείται από ακριβώς μία άτρακτο, δύο φτερά, δύο κινητήρες, ένα σύστημα τροχών και ένα σύστημα πλοήγησης. Κάθε φτερό αποτελείται από ένα δομικό τμήμα, από μία δεξαμενή καυσίμων και από ένα υδραυλικό σύστημα. Όπως καταλαβαίνει ο αναγνώστης, η ανάλυση μπορεί να συνεχίζεται για πάρα πολύ, φτάνοντας μέχρι και στις μικρότερες βίδες οι οποίες αποτελούν το αεροσκάφος.

### **Άλλες σχέσεις**

Μελετώντας την προτεινόμενη βιβλιογραφία μπορεί κανείς να βρει αναφορές και σε άλλες σχέσεις μεταξύ κλάσεων. Χαρακτηριστικά μπορούμε να αναφέρουμε τις σχέσεις «χρησιμοποιεί» (using), «στιγμιότυπο» (instantiation) και «μετακλάση» (meta-class). Οι σχέσεις αυτές αφορούν περισσότερο την υλοποίηση ενός συστήματος σε κάποια συγκεκριμένη γλώσσα προγραμματισμού που τις υποστηρίζει και δεν είναι συστατικά του μηχανισμού ανάλυσης προβλημάτων με την αντικειμενοστρεφή φιλοσοφία, ο οποίος θα μας απασχολήσει στο βιβλίο αυτό.

### **Άσκηση αυτοαξιολόγησης 2/Κεφάλαιο 7**

Ταξινομήστε τις παρακάτω οντότητες σε μια ιεραρχία, χρησιμοποιώντας τον μηχανισμό της κληρονομικότητας: δελφίνι, φάλαινα, ψάρι, θηλαστικό, αρκούδα, άνθρωπος, προγραμματιστής, εργαζόμενος, εκπαιδευτικός, ζωντανός οργανισμός, συνταξιούχος, εκπαιδευτικός πιάνου, πέστροφα, τσιπούρα.

## Άσκηση αυτοαξιολόγησης 3/Κεφάλαιο 7

Χαρακτηρίστε ως «Σωστό» ή «Λάθος» καθεμία από τις παρακάτω προτάσεις.

	Σωστό	Λάθος
1. Κληρονομικότητα και γενίκευση είναι δύο όψεις ενός νομίσματος.		
2. Η συσχέτιση (association) δεν είναι παρά μια άλλη ονομασία για τις σχέσεις του σχεσιακού μοντέλου δεδομένων.		
3. Κληρονομικότητα μπορούμε να έχουμε και στη δομημένη ανάλυση και σχεδίαση.		
4. Πολλαπλή κληρονομικότητα σημαίνει ότι μία κλάση αποδίδει τα χαρακτηριστικά της σε περισσότερες από μία κλάσεις-παιδιά.		
5. Η συσχέτιση (association) είναι ένα υπερσύνολο των σχέσεων μεταξύ πινάκων στις βάσεις δεδομένων.		
6. Ο ρόλος μιας κλάσης σε μια συσχέτιση εξαρτάται από τη φορά ανάγνωσης της συσχέτισης.		

## Δραστηριότητα 4/Κεφάλαιο 7

Προσπαθήστε να περιγράψετε με τη βοήθεια ενός διαγράμματος αντίστοιχου με εκείνο που φαίνεται στο Σχήμα 7.7 τη συναρμολόγηση ενός αυτοκινήτου. Να περιοριστείτε στα βασικά συστατικά και να μη φτάσετε σε πάνω από δύο επίπεδα λεπτομέρειας.

## Σύνοψη ενότητας

Στην αντικειμενοστρεφή τεχνολογία κεντρική είναι η έννοια της κλάσης. Κάθε κλάση περιέχει πεδία και μεθόδους, δηλαδή στοιχεία που αντιστοιχούν σε περιγραφή κατάστασης και συμπεριφοράς αντίστοιχα. Τα αντικείμενα είναι συστατικά στοιχεία λογισμικού που αποτελούν συγκεκριμένες εκδοχές μιας κλάσης, με δομή και συμπεριφορά που περιγράφονται πλήρως από την κλάση. Ιδιαίτερο χαρακτηριστικό της αντικειμενοστρεφούς τεχνολογίας είναι η κελυφοποίηση. Μεταξύ των

κλάσεων αναγνωρίζονται οι βασικές σχέσεις της συσχέτισης, της κληρονομικότητας ή γενίκευσης και της συναρμολόγησης.

## ΕΝΟΤΗΤΑ 7.6. ΕΝΑΣ ΑΛΛΟΣ ΤΡΟΠΟΣ ΠΑΡΑΣΤΑΣΗΣ ΤΟΥ ΚΟΣΜΟΥ

Έχοντας δει μια αρχική περιγραφή των χαρακτηριστικών της αντικειμενοστρεφούς προσέγγισης, μπορούμε τουλάχιστον να υποψιαστούμε ότι έχουμε στη διάθεσή μας όχι μόνο κάποιους νέους όρους που αφορούν τα συστατικά λογισμικού και τις σχέσεις μεταξύ τους αλλά και κάποια εργαλεία που μας επιτρέπουν να μοντελοποιήσουμε ένα πρόβλημα με διαφορετικό τρόπο απ' ό,τι η δομημένη ανάλυση και σχεδίαση. Πριν ξεκινήσουμε τη συζήτηση, είναι χρήσιμο να εισάγουμε την έννοια της αφαίρεσης (abstraction).

### Αφαίρεση:

Αφαίρεση (abstraction) είναι η νοητική εκείνη λειτουργία όπου από το σύνολο των λεπτομερειών μιας οντότητας ή ενός γεγονότος επιλέγουμε να επικεντρώσουμε την προσοχή μας μόνο σε κάποιες, αφαιρώντας τις υπόλοιπες.

Η αφαίρεση είναι χρήσιμο εργαλείο για τη δημιουργία μοντέλων της πραγματικότητας, τα οποία έχουν μόνο τη λεπτομέρεια που μας απασχολεί, και χρησιμοποιείται ευρύτατα στην αντικειμενοστρεφή προσέγγιση.

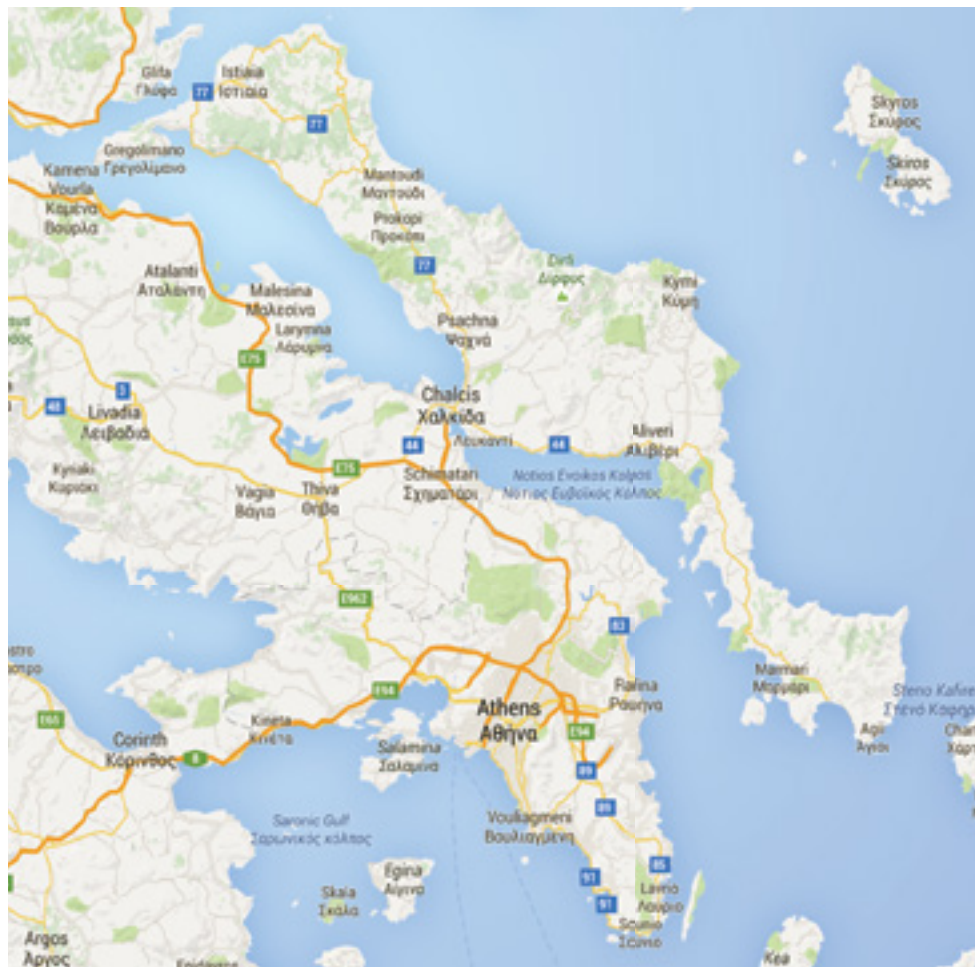
## Παράδειγμα 5/Κεφάλαιο 7

Ένας χάρτης είναι μια αφαίρεση της πραγματικότητας, η οποία απεικονίζει τα χαρακτηριστικά μιας γεωγραφικής περιοχής τα οποία μας ενδιαφέρουν. Ένας μορφολογικός χάρτης χρησιμοποιεί μια διαφορετική αφαίρεση απ' ό,τι ένας πολιτικός χάρτης της ίδιας περιοχής (Σχήμα 7.8). Δεν είναι εύκολο ούτε και χρήσιμο όλα τα χαρακτηριστικά της περιοχής να συμπεριληφθούν σε έναν και μόνο χάρτη. Για τον λόγο αυτό, κάνουμε τις αφαιρέσεις που κάθε στιγμή είναι χρήσιμες στον σκοπό μας και δουλεύουμε με αυτές.



**Σχήμα 7.8** Η έννοια της «αφαίρεσης». Στο σχήμα φαίνονται δύο εκδοχές του ίδιου αντικειμένου, εν προκειμένω ενός χάρτη. Η αφαίρεση στις δύο περιπτώσεις είναι διαφορετική, δηλαδή μας ενδιαφέρει να απεικονιστούν διαφορετικά χαρακτηριστικά.

Πηγή: Map data ©2015 Google





Θα προσπαθήσουμε ακολούθως να διαπραγματευτούμε σε συντομία το ερώτημα:

«Ποια είναι εκείνα τα χαρακτηριστικά της αντικειμενοστρεφούς τεχνολογίας που συνηγορούν στο ότι αυτή αντιμετωπίζει κάποιες από τις αδυναμίες της δομημένης ανάλυσης και σχεδίασης;»

Το πρώτο που μπορούμε να υποστηρίξουμε είναι ότι η ομαδοποίηση δεδομένων και λειτουργιών μέσα στις κλάσεις επιτρέπει την ευκολότερη αντιστοίχισή τους με οντότητες και έννοιες του πραγματικού κόσμου. Κάθε οντότητα του πραγματικού κόσμου έχει και κατάσταση και συμπεριφορά, και είναι περισσότερο κατανοητό να την παραστήσουμε με ένα σύνθετο αλλά ενιαίο συστατικό λογισμικού που ενσωματώνει και τα δύο, δηλαδή την κλάση, παρά με πολλά ανεξάρτητα μεταξύ τους.

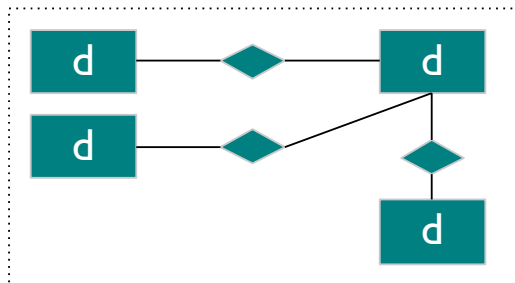
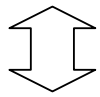
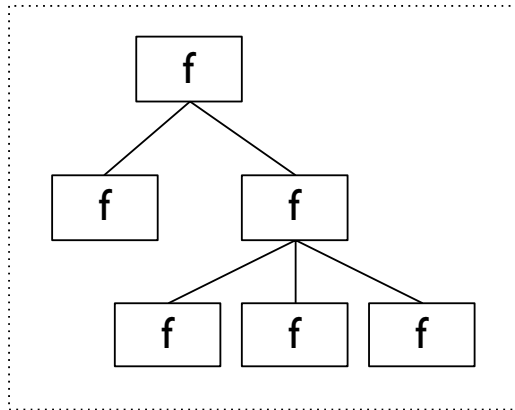
Δεύτερο σημείο είναι η δυνατότητα να αντιλαμβάνεται κανείς την ομαδοποίηση δεδομένων και λειτουργιών σε μια κλάση ως την ευθύνη της κλάσης να χειρίζεται δια των λειτουργιών της (δηλαδή να μεταβάλλει και να διαθέτει στο περιβάλλον της) τα δεδομένα που περικλείει. Η έννοια αυτή της ευθύνης αποτελεί κεντρικό σημείο σε αρκετές μεθοδολογίες αντικειμενοστρεφούς ανάλυσης.

Τρίτο σημείο είναι ο πλούτος των σχέσεων στην αντικειμενοστρεφή φιλοσοφία. Οι σχέσεις της κληρονομικότητας/γενίκευσης και της συναρμολόγησης δεν υπάρχουν καν ως έννοιες στη δομημένη ανάλυση. Μπορεί να τις συναντήσει κανείς να εφαρμόζονται ως πρακτικές στην κατασκευή του λογισμικού, μόνο που αυτό συμβαίνει χωρίς να αποτελούν εργαλείο μοντελοποίησης αλλά κατασκευαστικό εύρημα (για παράδειγμα, η παραμετροποίηση μιας συνάρτησης, ώστε να χρησιμοποιείται σε πολλές περιπτώσεις, μπορεί να θεωρηθεί ως μιας μορφής κληρονομικότητα).

Με λίγα λόγια, με την αντικειμενοστρεφή προσέγγιση κάνουμε ανάλυση του εκάστοτε προβλήματος σε διαφορετικά συστατικά απ' ό,τι με τη δομημένη. Η τοποθέτηση των δεδομένων (πεδία) μαζί με τις λειτουργίες που επιδρούν σε αυτά (μεθόδους) αναδεικνύει τα δεδομένα σε κυρίαρχα στοιχεία της αναλυτικής σκέψης στην αντικειμενοστρεφή φιλοσοφία (Σχήμα 7.9).

**Σχήμα 7.9** Αφαίρεση ως προς τους μετασχηματισμούς και ως προς τα δεδομένα.

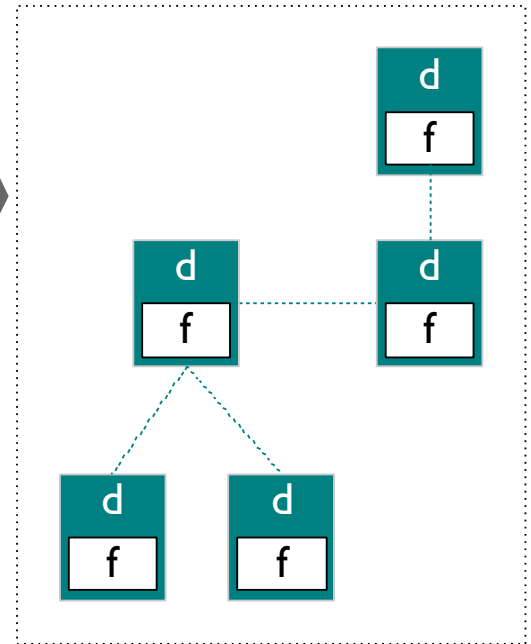
### Functional abstraction



Μοντελοποίηση με  
λειτουργίες και  
ανεξάρτητα δεδομένα.

Πρόβλημα του  
πραγματικού  
κόσμου

### Data abstraction



Μοντελοποίηση με  
δεδομένα ενοποιημένα  
με τις λειτουργίες που  
τα αφορούν.

Στην αντικειμενοστρεφή φιλοσοφία ακολουθούμε αφαίρεση ως προς τα δεδομένα (data abstraction) και μαζί με αυτά τοποθετούμε τις λειτουργίες, πράγμα που δεν ισχύει στη δομημένη ανάλυση όπου ακολουθούμε αφαίρεση ως προς τους μετασχηματισμούς (functional abstraction), οι οποίοι τελικά είναι μονάδες προγράμματος που επιδρούν σε εντελώς ανεξάρτητα δεδομένα.

Το αριστερό μέρος του σχήματος φαίνεται αρκετά οικείο: ένα πρόγραμμα που κατασκευάστηκε ακολουθώντας τη δομημένη φιλοσοφία αποτελείται από ένα σύνολο ενεργών συστατικών λογισμικού (λ.χ. συναρτήσεων) τα οποία επιδρούν πάνω σε ανεξάρτητα δεδομένα. Η ενεργοποίηση των λειτουργιών γίνεται με τη χρήση κλήσεων (calls) μεταξύ των ενεργών συστατικών του λογισμικού.

Στο δεξί μέρος του σχήματος έχουμε μια διαφορετική εικόνα: ένα πρόγραμμα κατασκευασμένο σύμφωνα με την αντικειμενοστρεφή φιλοσοφία είναι ένα σύνολο από αντικείμενα, τα οποία γίνονται με δυναμικό τρόπο ενεργά στο πεδίο της εκτέλεσης (runtime). Τα αντικείμενα αυτά προσφέρουν το ένα στο άλλο τις υπηρεσίες για τις οποίες είναι υπεύθυνα, εκδηλώνοντας με τον τρόπο αυτό τη λειτουργική τους συμπεριφορά. Στο πεδίο ευθύνης των αντικειμένων, εκτός από την εκδήλωση της συμπεριφοράς αυτής, ανήκει και η διαχείριση κάποιων δεδομένων τα οποία παύουν πλέον να είναι ανεξάρτητα.

Κλείνοντας την ενότητα αυτή, είναι χρήσιμο να σημειώσουμε ότι η αντικειμενοστρεφής προσέγγιση δεν είναι εξ ορισμού η καλύτερη για όλα τα προβλήματα. Απλώς, την εποχή που διανύουμε τη θεωρούμε ως την πιο πρόσφορη για το είδος των υπολογιστικών προβλημάτων που αντιμετωπίζουμε. Όπως αναφέραμε στην αρχή της ενότητας, η ιδέα υπήρχε από τη δεκαετία του 1960. Τότε όμως επικρατούσε η άποψη ότι τα προβλήματα στην ανάπτυξη του λογισμικού θα αντιμετωπιστούν με τη δομημένη ανάλυση και σχεδίαση. Ήταν, αναμφίβολα, και το είδος των εφαρμογών λογισμικού εκείνης της εποχής που συνέβαλλε στην υποστήριξη αυτής της θέσης.

Αύριο, το είδος των εφαρμογών λογισμικού που θα αναπτύσσουμε μπορεί να μας οδηγήσει στην άποψη ότι ο λογικός προγραμματισμός (logic programming) είναι πιο πρόσφορος για την αντιμετώπιση των νέων αναγκών στην ανάπτυξη λογισμικού (η ιδέα υπάρχει ήδη από δεκαετίες). Σήμερα, τόσο οι εξελίξεις που αναφέραμε, όσο και το είδος των εφαρμογών λογισμικού που κατασκευάζουμε, μας ωθούν στο να πιστεύουμε ότι η αντικειμενοστρεφής προσέγγιση είναι πιο πρόσφορη και προσπαθούμε να την κάνουμε επαρκώς γενική και εφαρμόσιμη για τους σκοπούς μας.

## Σύνοψη ενότητας

---

*Η αντικειμενοστρεφής τεχνολογία, εκτός από τις νέες έννοιες που εισάγει σε κατασκευαστικό επίπεδο, προσφέρει πρώτα και κύρια έναν άλλο τρόπο ανάλυσης των προβλημάτων του πραγματικού κόσμου και παράστασης της λύσης τους με χρήση προγράμματος ηλεκτρονικού υπολογιστή. Πρόκειται για ένα άλλο μεθοδολογικό εργαλείο σκέψης και όχι μόνο για ένα νέο σύνολο δομών στον προγραμματισμό, το οποίο χαρακτηρίζεται από την αφαίρεση ως προς τα δεδομένα (data abstraction), ενώ η δομημένη προσέγγιση χαρακτηρίζεται από αφαίρεση ως προς τους μετασχηματισμούς (functional abstraction).*

## ΕΝΟΤΗΤΑ 7.7. ΣΥΜΒΟΛΙΣΜΟΙ ΚΑΙ ΠΡΟΤΥΠΑ

Επί μακρόν, στην κοινότητα της αντικειμενοστρεφούς τεχνολογίας επικρατούσε μια σύγχυση και ένας πλουραλισμός συμβολισμών για τις διάφορες έννοιες. Αιτίες γι' αυτό μπορούν να αναζητηθούν σε πολλά επίπεδα, όπως ενδεικτικά:

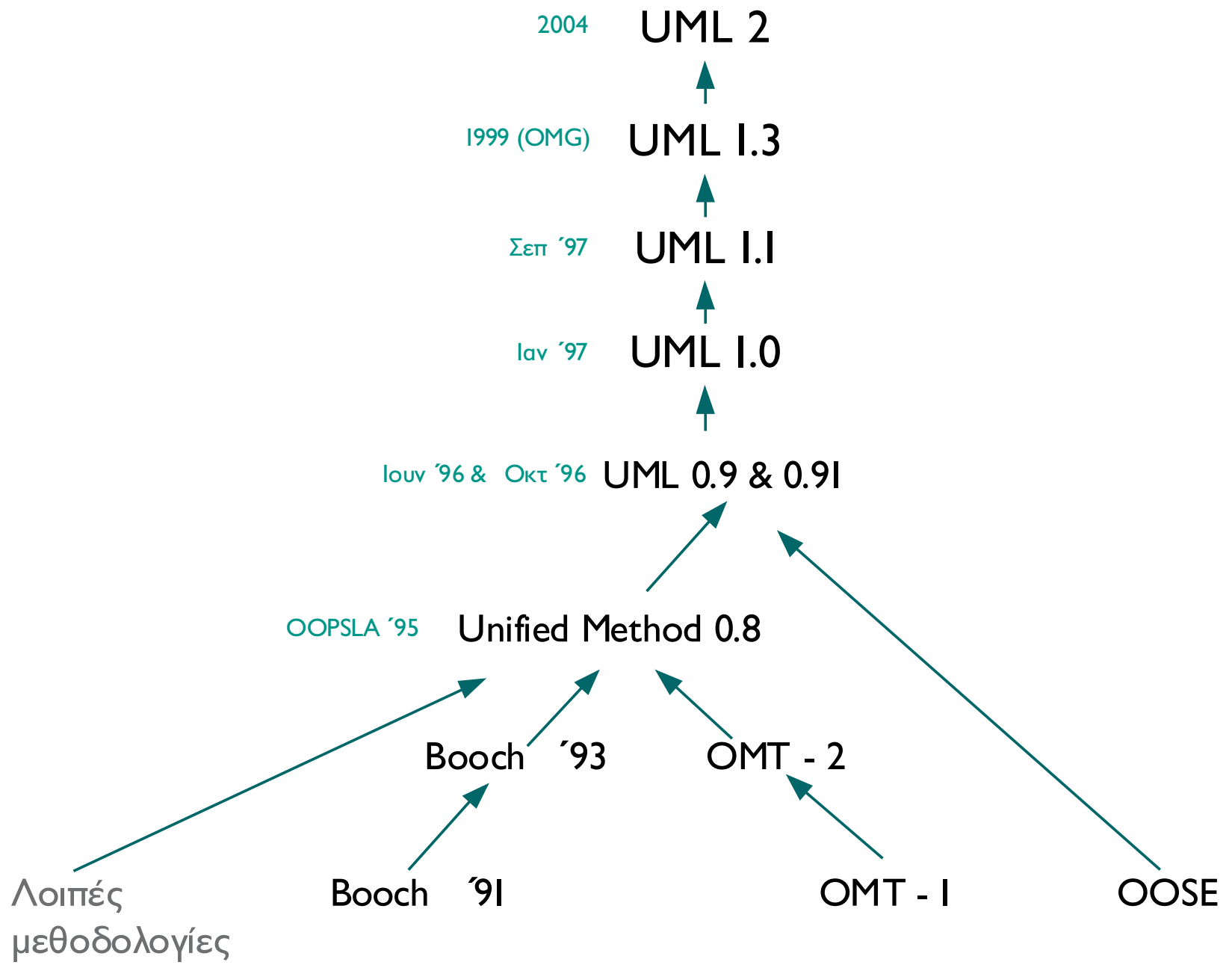
- Το σχετικά νέο της εμφάνισης της αντικειμενοστρεφούς τεχνολογίας στους κατασκευαστές λογισμικού.
- Η ρευστότητα των εξελίξεων και η μη επικράτηση καμίας προσέγγισης.
- Ο ανταγωνισμός στην αγορά εργαλείων και συμβουλευτικών υπηρεσιών ανάπτυξης λογισμικού.

- Η προερχόμενη από τη δομημένη φιλοσοφία κουλτούρα πολλών συγγραφέων και ερευνητών, καθώς και των περισσότερων κατασκευαστών λογισμικού.
- Η σχετικά περιορισμένη (την εποχή εκείνη) ευρύτητα του πεδίου εφαρμογών των οποίων η ανάπτυξη με αντικειμενοστρεφή λογική θα ήταν πιο πρόσφορη.

Με την ωρίμανση της τεχνολογίας, όπως αναφέρθηκε και στο πρώτο κεφάλαιο του βιβλίου αυτού, δημιουργήθηκαν οι συνθήκες για την ανάπτυξη ενός προτύπου συμβολισμών των εννοιών που είχαν σχέση με την αντικειμενοστρεφή τεχνολογία. Τα πρώτα βήματα έγιναν στις αρχές της δεκαετίας του 1990 από τους συγγραφείς των τριών επικρατέστερων αντικειμενοστρεφών προσεγγίσεων ανάπτυξης λογισμικού: τους Booch, Jacobson και Rumbaugh (προφέρεται «Ράμπο»!).

Αργότερα, αναγνωρίζοντας τη δυναμική που δημιουργείται, συνέπραξαν και μεγάλοι κατασκευαστές λογισμικού και εργαλείων, όπως η Digital Equipment (τότε υπήρχε ακόμη), η Hewlett Packard, η Oracle, η Texas Instruments, η Unisys, η MCI και η Microsoft. Ακολουθώντας την πορεία που φαίνεται στο Σχήμα 7.10, κατέληξαν στην περιγραφή ενός προτύπου συμβολισμών το οποίο ονομάστηκε Unified Modeling Language και εν συντομία UML. Το πρότυπο, υιοθετήθηκε από τον οργανισμό OMG (Object Management Group) ως βιομηχανικό πρότυπο παράστασης λογισμικού.

### Σχήμα 7.10 Η πορεία προς την UML.



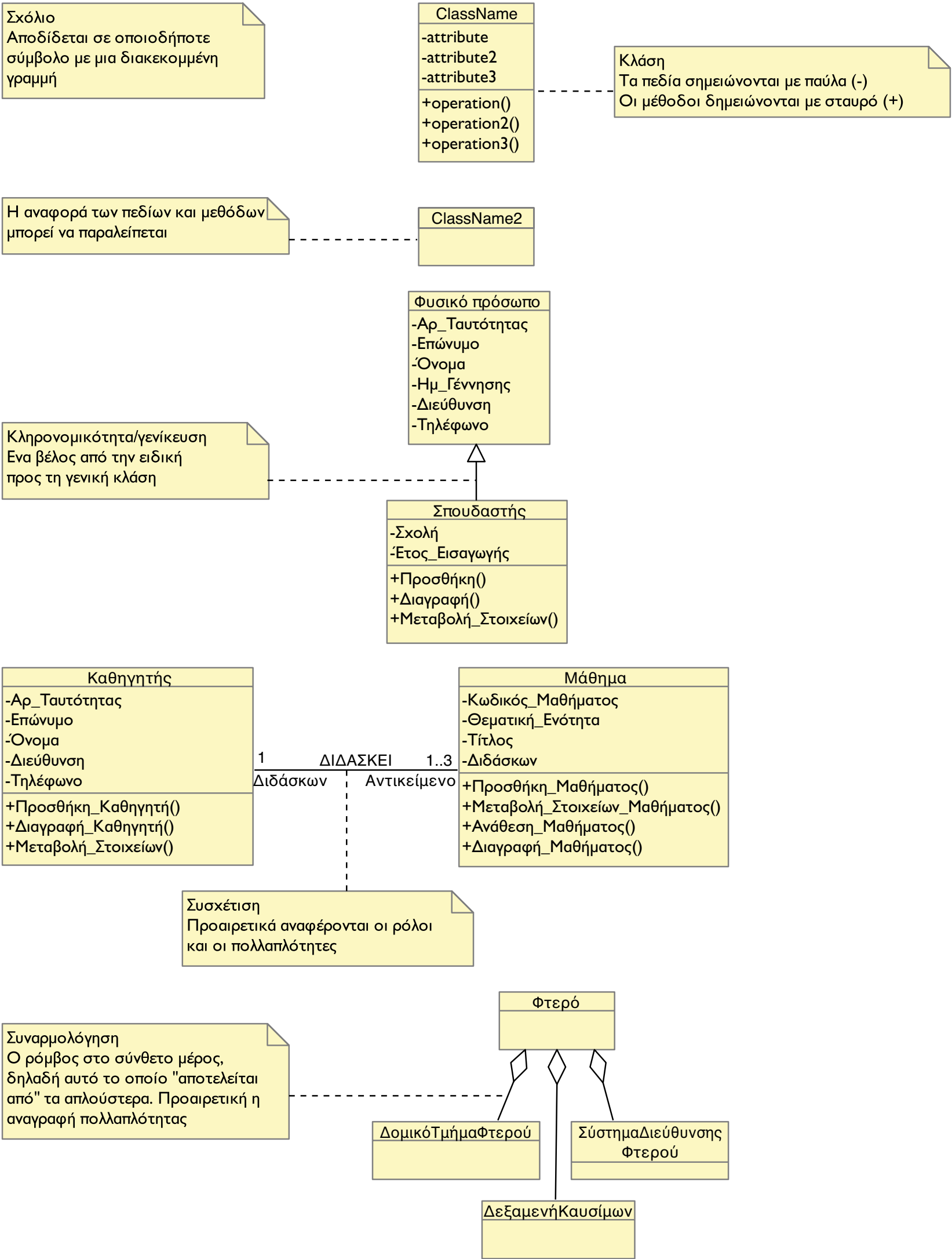
Η UML από μόνη της μπορεί να αποτελέσει αυτοτελές αντικείμενο μακράς ενασχόλησης, κάτι που ξεφεύγει από το αντικείμενο του παρόντος βιβλίου. Είναι κατάλληλη για την παράσταση των εννοιών που απαντώνται όχι μόνο στην αντικειμενοστρεφή τεχνολογία ανάπτυξης λογισμικού αλλά και σε αρκετά άλλα θεματικά πεδία, όπως η μοντελοποίηση επιχειρησιακών διαδικασιών (Business Modeling), προσφέροντας ένα πρότυπο σύνολο συμβολισμών το οποίο τελικά αποτελεί μέσο επικοινωνίας.

Ως τέτοιο θα την αντιμετωπίσουμε στο βιβλίο αυτό, χρησιμοποιώντας τα συστατικά που η ίδια διαθέτει προκειμένου να την ορίσουμε, πράγμα που θα κάνουμε διαδοχικά, σε σημεία όπου θα είναι αναγκαίο να χρησιμοποιήσουμε κάποιο στοιχείο της (σύμβολο ή διάγραμμα). Με αυτό τον τρόπο θα επιδιώξουμε να επαληθεύσουμε την καταλληλότητά της ως εργαλείο επικοινωνίας και κοινό σημείο αναφοράς συμβολισμών.

## **Συμβολισμοί UML**

Στο Σχήμα 7.11 φαίνονται ορισμένα από τα βασικά σύμβολα της UML, τα οποία σχετίζονται με τις έννοιες που παρουσιάσαμε μέχρι το σημείο αυτό.

Σχήμα 7.11 Συμβολισμοί UML.





## ΕΝΟΤΗΤΑ 7.8. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ

### Δραστηριότητα 1/Κεφάλαιο 7

---

Επιγραμματικά, οι σημαντικότερες εξελίξεις είναι οι ακόλουθες:

- Απομυθοποιήθηκαν οι αυστηρές και αναποτελεσματικές διαδικασίες ανάπτυξης λογισμικού.
- Το υλικό των υπολογιστών σημείωσε εντυπωσιακή πρόοδο και απόκτησε ιδιαίτερα υψηλές δυνατότητες.
- Το διαδίκτυο εξελίχθηκε σε πλατφόρμα λειτουργίας λογισμικού και σημειώθηκε μεγάλη ζήτηση εφαρμογών.
- Ξεκαθάρισε κάπως το τοπίο με τη σύγκλιση των σημαντικότερων αντικειμενοστρεφών προσεγγίσεων ανάπτυξης λογισμικού.

Οι σημαντικότερες τάσεις, επίσης επιγραμματικά, είναι οι ακόλουθες:

- Τα μοντέλα κύκλου ζωής λογισμικού έγιναν πιο ρεαλιστικά και προσαρμόσιμα.
- Η τεκμηρίωση λογισμικού έγινε πιο ελαστική.
- Τα περιβάλλοντα ανάπτυξης λογισμικού έγιναν πληρέστερα.
- Τα εργαλεία υποστήριξης της ανάπτυξης λογισμικού ωρίμασαν.

Μπορείτε, ασφαλώς, να δώσετε και με άλλον τρόπο την επιγραμματική αυτή απόδοση. Αυτό που έχει σημασία είναι ότι δεν πρόκειται για άσκηση περίληψης κειμένου αλλά για προσπάθεια κατανόησης σύνθετων τρεχουσών εξελίξεων, οι οποίες δεν είναι πάντα ορατές. Συμπληρώστε, αν θέλετε, τις εξελίξεις και τις τάσεις αναλαμβάνοντας, όπως και εμείς, το ρίσκο να πέσετε έξω.

### Δραστηριότητα 2/Κεφάλαιο 7

---

Όπως είδαμε, υπάρχουν οι ακόλουθες έκδηλες, πλέον, δυσκολίες στην ανάπτυξη λογισμικού με τη δομημένη ανάλυση και σχεδίαση:

**Στο θεωρητικό επίπεδο:**

I. Η δομημένη ανάλυση και σχεδίαση υποτιμά τα δεδομένα.

2. Οι συναρτήσεις δεν υπάρχουν ανεξάρτητα από τα δεδομένα στον πραγματικό κόσμο.

### **Στο πρακτικό επίπεδο:**

1. Ο προσδιορισμός των απαιτήσεων σε όρους μετασχηματισμών και ανεξάρτητων δεδομένων είναι εξαιρετικά δύσκολος.
2. Η διαχείριση των μοντέλων παράστασης λογισμικού που χρησιμοποιεί η δομημένη φιλοσοφία είναι πολύ δύσκολη.
3. Η συντήρηση του λογισμικού είναι δύσκολη και δαπανηρή.
4. Η επαναχρησιμοποίηση συστατικών λογισμικού δεν ενθαρρύνεται.

Μπορείτε εύκολα να συνηγορήσετε υπέρ των δυσκολιών αυτών και, μάλιστα, όσο περισσότερο έχετε ασχοληθεί με την ανάπτυξη λογισμικού, τόσο περισσότερο μπορείτε να τις τονίσετε. Ευχόμαστε μετά την ενασχόλησή σας με την αντικειμενοστρεφή προσέγγιση να εκτιμάτε ότι σε κάποιες από αυτές τις δυσκολίες η αντικειμενοστρεφής προσέγγιση προσφέρει λύσεις.

## **Δραστηριότητα 3/Κεφάλαιο 7**

---

Η κλάση «σπουδαστής» θα έχει (τουλάχιστον) τα πεδία που αναφέρονται στο λεξικό δεδομένων της μελέτης περίπτωσης του πρώτου τόμου. Σε αναλογία με τις κλάσεις του Παραδείγματος Ι, θα της αποδώσουμε μεθόδους για την προσθήκη, μεταβολή και διαγραφή στοιχείων σπουδαστή. Μαζί με ένα αντικείμενο, το οποίο αντιστοιχεί σε κάποιο συγκεκριμένο σπουδαστή, η ζητούμενη κλάση φαίνεται στο Σχήμα 7.12.

**Σχήμα 7.12** Η κλάση «σπουδαστής» μαζί με ένα αντικείμενο αυτής.

Student
-ID_no -FirstName -LastName -Address -Phone -Department
+NewStudent() +DeleteStudent() +ModifyStudent()

Student001
-ID_no: A123456 -FirstName: Νικόλαος -LastName: Νικολάου -Address: Αγ.Νικολάου 1 -Phone: 2101234567 -Department: Πληροφορικής
+NewStudent() +DeleteStudent() +ModifyStudent()

## ΒΙΒΛΙΟΓΡΑΦΙΑ

*Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example*, Doug Rosenberg και Kendall Scott, Addison-Wesley, ISBN 978-0201730395

Booch G., Rumbaugh J., Jacobson I., *The Unified Modeling Language User Guide*, Addison-Wesley.

Henderson-Sellers B., *A Book of Object-Oriented Knowledge*, Prentice Hall.

Khoshafian S., Abnous R., *Object Orientation: Concepts, Languages, Databases, User Interfaces*, Wiley.

Martin J., Odell J., *Object-Oriented Analysis and Design*, Prentice Hall.

Pressman, R. S., *Software Engineering-A Practitioners Approach*, McGraw-Hill.

Rumbaugh J., Jacobson I., Booch G., *The Unified Modeling Language Reference Manual*, Addison-Wesley.

Sommerville, I. *Software Engineering*, London: Addison-Wesley.

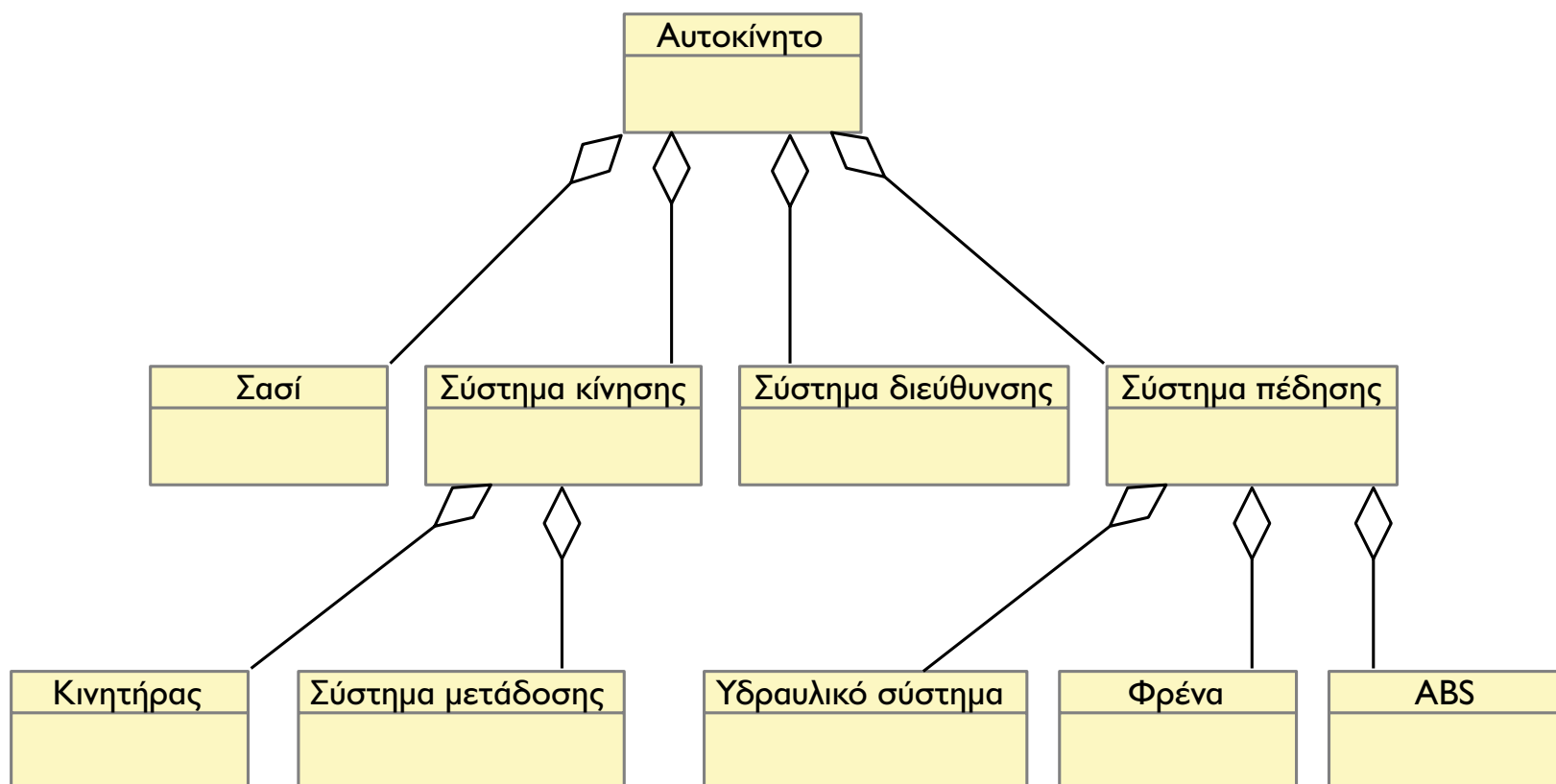
Μην ανησυχείτε ούτε για την απόδοση πεδίων και μεθόδων σε κλάσεις ούτε για τους συμβολισμούς. Όλα θα εξηγηθούν στη συνέχεια. Αυτό που έχει αυτήν τη στιγμή σημασία είναι αποκτήσουμε μια πρώτη εικόνα για την έννοια. Συγχαρητήρια σε όσους έδωσαν άμεσα μια απάντηση όπως αυτή που δίνεται στο Σχήμα 7.12 ή και πληρέστερη.

## **Δραστηριότητα 4/Κεφάλαιο 7**

---

Στο Σχήμα 7.13 φαίνεται μια ανάλυση ενός αυτοκινήτου σε επιμέρους συστατικά. Όπου δεν σημειώνεται η πολλαπλότητα, υπονοείται ότι αυτή είναι 1.

**Σχήμα 7.13** Ενδεικτική αποσυναρμολόγηση ενός αυτοκινήτου σε επιμέρους τμήματα.



Όσοι έχουν ιδιαίτερη σχέση με τα αυτοκίνητα σίγουρα θεώρησαν τη δραστηριότητα παιχνίδι (αν και ελπίζουμε να μην το παρακάνετε!). Για τους υπόλοιπους, μια ματιά σε ένα παρκαρισμένο αυτοκίνητο και σε κάποιες διαφημίσεις αυτοκινήτων είναι πιθανότατα επαρκής για να προσεγγίσουν το θέμα. Εξάλλου, πρόκειται για άσκηση λογισμικού και όχι... μηχανολογίας.

### Άσκηση Ι/Κεφάλαιο 7

	Σωστό	Λάθος
<b>1. Μια κλάση περιέχει αντικείμενα.</b> Μια κλάση περιέχει πεδία και μεθόδους. Τα αντικείμενα είναι συγκεκριμένες εκδοχές της κλάσης.		●
<b>2. Μια κλάση περιέχει μεθόδους και πεδία.</b>	●	
<b>3. Μια κλάση είναι το στιγμιότυπο των αντικειμένων της.</b> Κάθε αντικείμενο είναι στιγμιότυπο της κλάσης.		●
<b>4. Κάθε μέθοδος ορίζεται στον ορισμό του αντικειμένου όπου ανήκει.</b> Κάθε μέθοδος ορίζεται με τον ορισμό της κλάσης.		●
<b>5. Οι μέθοδοι αντιστοιχούν στην κατάσταση μιας κλάσης.</b> Οι μέθοδοι περιγράφουν τη συμπεριφορά μιας κλάσης ή, ακριβέστερα, των αντικειμένων – στιγμιοτύπων αυτής.		●
<b>6. Οι τιμές των πεδίων περιγράφουν την κατάσταση ενός αντικειμένου.</b>	●	
<b>7. Η συμπεριφορά όλων των αντικειμένων μιας κλάσης είναι ίδια.</b>	●	
<b>8. Η μέθοδος είναι μια συνάρτηση ή διαδικασία ενσωματωμένη μέσα σε μια κλάση.</b>	●	
<b>9. Αντικείμενο = (εγγραφή) + (συναρτήσεις/διαδικασίες).</b>	●	
<b>10. Ένα αντικείμενο χαρακτηρίζεται από κατάσταση και συμπεριφορά.</b> Κάθε αντικείμενο χαρακτηρίζεται από κατάσταση, συμπεριφορά και ταυτότητα.		●

<b>II. Η έννοια της κλάσης είναι αντίστοιχη της έννοι- ας της μεταβλητής μνήμης.</b> Η έννοια του αντικειμένου είναι αντίστοιχη της μεταβλη- τής. Η κλάση είναι αντίστοιχη του τύπου.		●
<b>I2. Η απόκρυψη πληροφοριών σχετίζεται με την το- ποθέτηση μεθόδων και πεδίων μέσα στο κέλυφος μιας κλάσης.</b> Σχετίζεται με την ελεγχόμενη πρόσβαση σε αυτά και όχι μόνο με την ύπαρξή τους μέσα στο κέλυφος της κλάσης		●

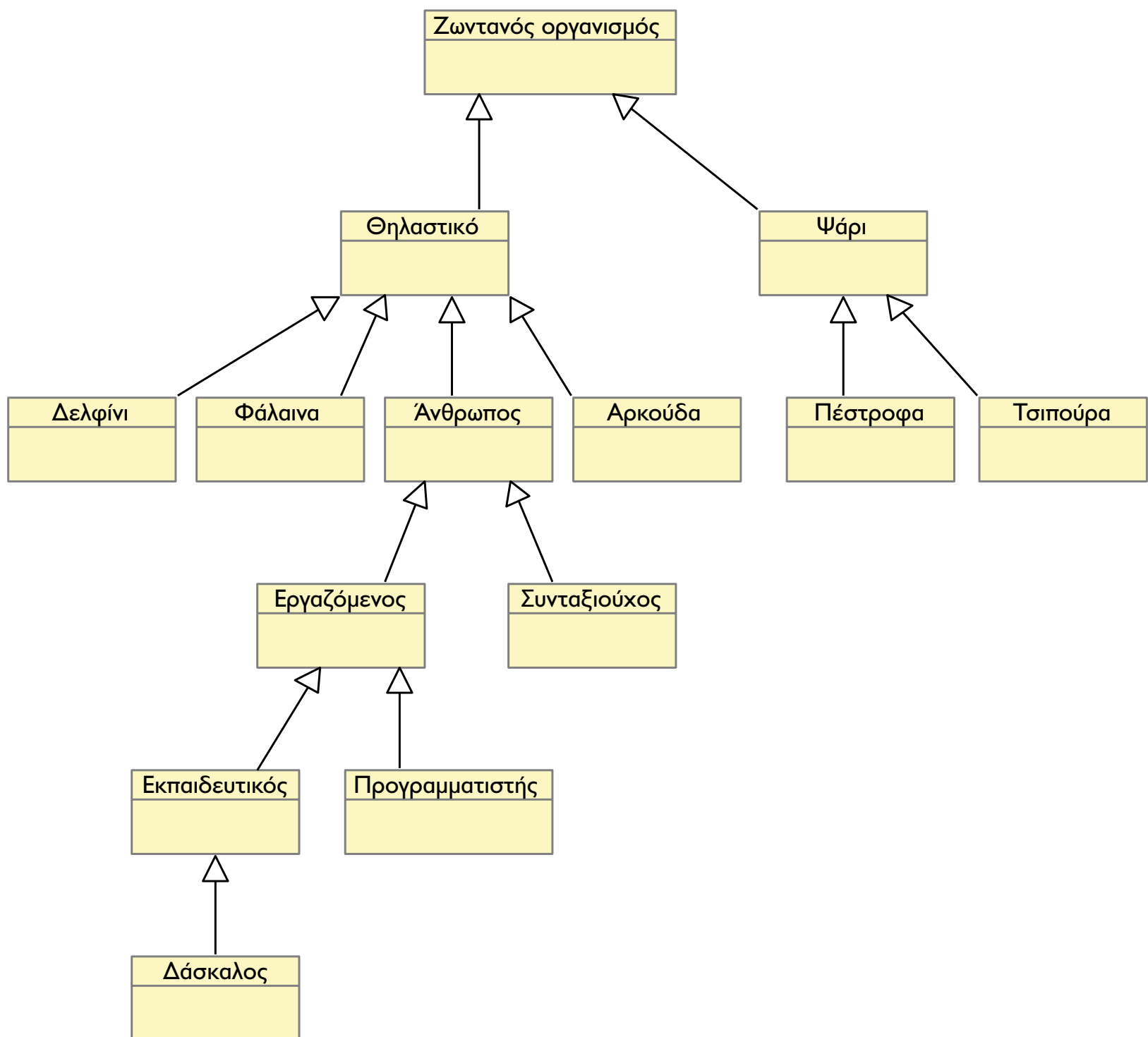
Όσοι δώσατε τις σωστές απαντήσεις με την πρώτη αξίζετε συγχαρητήρια, αλλά να θυμάστε πως η προσπάθεια μόλις ξεκίνησε.

### Άσκηση 2/Κεφάλαιο 7

Στο Σχήμα 7.14 δίνεται η ζητούμενη ταξινόμηση.



**Σχήμα 7.14** Ταξινόμηση ζωντανών οργανισμών.



Πρόκειται για δουλειά που απαιτεί ελάχιστες γνώσεις, ωστόσο ίσως κάποιοι να κάνουν το σφάλμα να χαρακτηρίσετε τη φάλαινα και το δελφίνι ως ψάρια, ενώ είναι θηλαστικά.

## Άσκηση 3/Κεφάλαιο 7

	Σωστό	Λάθος
<b>1. Κληρονομικότητα και γενίκευση είναι δύο όψεις ενός νομίσματος.</b>	●	
<b>2. Η συσχέτιση (association) δεν είναι παρά μια άλλη ονομασία για τις σχέσεις του σχεσιακού μοντέλου δεδομένων.</b> Η συσχέτιση είναι μια γενίκευση των σχέσεων του σχεσιακού μοντέλου δεδομένων.		●
<b>3. Κληρονομικότητα μπορούμε να έχουμε και στη δομημένη ανάλυση και σχεδίαση.</b> Μπορούμε να χρησιμοποιήσουμε προγραμματιστικά τεχνάσματα ώστε να βγάλουμε «κοινό παράγοντα» από κάποιες συναρτήσεις ή διαδικασίες, αλλά αυτό δεν είναι κληρονομικότητα.		●
<b>4. Πολλαπλή κληρονομικότητα σημαίνει ότι μία κλάση αποδίδει τα χαρακτηριστικά της σε περισσότερες από μία κλάσεις-παιδιά.</b> Σημαίνει ότι μια κλάση κληρονομεί χαρακτηριστικά από δύο ή περισσότερες κλάσεις-πατέρες.		●
<b>5. Η συσχέτιση (association) είναι ένα υπερσύνολο των σχέσεων μεταξύ πινάκων στις βάσεις δεδομένων.</b>	●	
<b>6. Ο ρόλος μιας κλάσης σε μια συσχέτιση εξαρτάται από τη φορά ανάγνωσης της συσχέτισης.</b> Η ονομασία της συσχέτισης μπορεί να διαφέρει ανάλογα με τη φορά ανάγνωσης της σχέσης και όχι ο ρόλος κάθε κλάσης σε αυτή.		●

Η σωστή απάντηση με την πρώτη σάς φέρνει είτε στην πρώτη γραμμή της προσέγγισης της αντικειμενοστρεφούς τεχνολογίας είτε ανάμεσα στους πιο τυχερούς.

## ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗ ΑΝΑΛΥΣΗ

Σκοπός του κεφαλαίου είναι να εισάγει τον αναγνώστη στην αντικειμενοστρεφή ανάλυση σύμφωνα με την ενοποιημένη προσέγγιση ανάπτυξης λογισμικού, η οποία αποτελεί το προϊόν σύγκλισης τριών από τις επικρατέστερες προσεγγίσεις ανάπτυξης λογισμικού με την αντικειμενοστρεφή φιλοσοφία.

Μετά τη μελέτη του κεφαλαίου αυτού, ο αναγνώστης θα είναι σε θέση:

- να αναφέρει τέσσερα χαρακτηριστικά της ενοποιημένης προσέγγισης ανάπτυξης λογισμικού και να περιγράφει το μοντέλο κύκλου ζωής που αυτή ακολουθεί,
- να καταγράφει τις λειτουργικές απαιτήσεις από το λογισμικό ως ένα σύνολο από περιπτώσεις χρήσης και το περιβάλλον λειτουργίας μιας εφαρμογής λογισμικού ως σύνολο από χειριστές,
- να χρησιμοποιεί τεχνικές αντικειμενοστρεφούς ανάλυσης, ώστε να κατασκευάζει το μοντέλο ανάλυσης μιας εφαρμογής λογισμικού σύμφωνα με την ενοποιημένη προσέγγιση ανάπτυξης λογισμικού,
- να ορίζει κλάσεις ανάλυσης από τις περιπτώσεις χρήσης, να τις διακρίνει σε τρεις κατηγορίες και να χρησιμοποιεί τα σύμβολά τους στο διάγραμμα κλάσεων,
- να εντοπίζει τις ευθύνες και τις συσχετίσεις μεταξύ των κλάσεων στο μοντέλο ανάλυσης,
- να χρησιμοποιεί τα διαγράμματα περιπτώσεων χρήσης, δραστηριοτήτων και συνεργασίας, καθώς και δομημένο κείμενο για να τεκμηριώνει το μοντέλο ανάλυσης.

## Έννοιες-κλειδιά

---

- Αντικειμενοστρεφής ανάλυση
- Ενοποιημένη προσέγγιση
- Περίπτωση χρήσης
- Χειριστής
- Διάγραμμα περιπτώσεων χρήσης
- Διάγραμμα δραστηριοτήτων
- Διάγραμμα συνεργασίας • Κλάση οντοτήτων
- Συνοριακή κλάση
- Κλάση ελέγχου
- Πακέτο ανάλυσης
- Μοντέλο ανάλυσης
- Μοντέλο περιπτώσεων χρήσης

## Σύνοψη

---

Η ενοποιημένη προσέγγιση ακολουθεί το γενικό μοντέλο κύκλου ζωής λογισμικού και η ανάπτυξη λογισμικού, σύμφωνα με αυτή, χαρακτηρίζεται ως επαναληπτική και επαυξητική. Στην αντικειμενοστρεφή ανάλυση, σύμφωνα με την ενοποιημένη προσέγγιση, οι λειτουργικές απαιτήσεις παριστάνονται ως περιπτώσεις χρήσης. Οι περιπτώσεις χρήσης είναι μια κεντρική έννοια στην ενοποιημένη προσέγγιση και όλα τα προϊόντα και οι εργασίες της ανάπτυξης λογισμικού αναφέρονται σε αυτές. Κάθε περίπτωση χρήσης τεκμηριώνεται με τη βοήθεια δομημένου κειμένου και διαγραμμάτων. Η τεκμηρίωση αυτή αποτελεί την είσοδο στην εργασία της ανάλυσης, όπου το λογισμικό διασπάται σε τμήματα (πακέτα) στα οποία αντιστοιχίζονται οι περιπτώσεις χρήσης. Για κάθε περίπτωση χρήσης εντοπίζονται οι κλάσεις και η απαιτούμενη συνεργασία αυτών, ώστε να παραχθεί το επιθυμητό αποτέλεσμα. Στην ανάλυση διακρίνονται τρεις τύποι κλάσεων: οι κλάσεις οντοτήτων, οι συνοριακές κλάσεις και οι κλάσεις ελέγχου. Τέλος, προσδιορίζονται τα πεδία και οι συσχετίσεις

μεταξύ των κλάσεων και επαληθεύεται η δόμηση του μοντέλου ανάλυσης, καθώς και οι εξαρτήσεις των πακέτων.

Οι εργασίες της ανάλυσης δεν αφορούν ολόκληρη την εφαρμογή λογισμικού αλλά μόνο το τμήμα αυτής με το οποίο ο κατασκευαστής επιλέγει να ασχοληθεί σε έναν κύκλο ανάπτυξης. Με αυτόν τον τρόπο γίνεται ευκολότερη η διαχείριση της ανάπτυξης και ελαχιστοποιείται το ρίσκο αλλά και οι επιπτώσεις εσφαλμένων επιλογών. Η προσοχή κατά την ανάλυση εστιάζεται στα ποιοτικά χαρακτηριστικά της υλοποίησης των ροών των περιπτώσεων χρήσης και όχι σε κατασκευαστικές λεπτομέρειες, μη λειτουργικές απαιτήσεις και βελτιστοποιήσεις, προβλήματα με τα οποία καταπιάνεται η σχεδίαση.

Σε πολλές περιπτώσεις είναι αναπόφευκτη η σύγκριση της δομημένης με την αντικειμενοστρεφή φιλοσοφία, όχι μόνο για να τεκμηριωθεί ποια είναι καλύτερη αλλά και για να γίνει πιο συνειδητή η δεύτερη ως νεότερη. Σε τελική ανάλυση, δηλαδή στο επίπεδο αυτού καθεαυτού του πηγαίου κώδικα όπου καταλήγει οποιαδήποτε προσέγγιση ανάλυσης και σχεδίασης, είναι ο δομημένος προγραμματισμός που χρησιμοποιείται. Η δομημένη φιλοσοφία χρησιμοποιεί την έννοια της ιεραρχικής δομής τόσο στο μακρο-επίπεδο (αρχιτεκτονική) όσο και στο μικρο-επίπεδο (πηγαίος κώδικας) του λογισμικού. Η αντικειμενοστρεφής αντιμετωπίζει με άλλο τρόπο το μακρο-επίπεδο, χρησιμοποιεί όμως –τοποθετημένο μέσα στο δικό της κέλυφος– τον δομημένο πηγαίο κώδικα. Μπορείτε να περιπλανηθείτε στη βιβλιογραφία όπου θα βρείτε εκτενείς συγκριτικές αναφορές, όχι απαραίτητα όλες συγκλίνουσες.

## Εισαγωγικές παρατηρήσεις

---

Η εισαγωγή της αντικειμενοστρεφούς τεχνολογίας στην κοινότητα των κατασκευαστών λογισμικού σηματοδοτήθηκε από πλουραλισμό απόψεων σχετικά με το πώς θα εφαρμόζονταν στην πράξη οι γενικές αρχές αυτής. Ακόμη και σήμερα δεν μπορούμε να μιλάμε για μια γενικώς αποδεκτή και εφαρμόσιμη προσέγγιση ούτε καν για τα γενικά χαρακτηριστικά μιας οικογένειας προσεγγίσεων, όπως κάναμε στην περίπτωση της δομημένης ανάλυσης.

Το γεγονός αυτό είναι ενδεικτικό της ρευστότητας των πραγμάτων και μας αναγκάζει να επιλέξουμε να αναφερθούμε σε μία από τις υπάρχουσες σήμερα προσεγγίσεις αντικειμενοστρεφούς ανάλυσης και σχεδίασης. Πρόκειται για το αποτέλεσμα

της συγχώνευσης τριών από τις επικρατέστερες προσεγγίσεις και, συγκεκριμένα, των Jacobson, Booch και Rumbaugh (προφέρεται «ράμπο»), οι οποίες περιλαμβάνονται στην βιβλιογραφία. Η σύγκλιση ξεκίνησε μέσα στο 1997 με την προσπάθεια αναζήτησης ενός προτύπου συμβολισμών, η οποία κατέληξε στην UML, και ολοκληρώθηκε περίπου δύο χρόνια αργότερα με την παρουσίαση μιας ενοποιημένης προσέγγισης ανάπτυξης λογισμικού η οποία συγκέντρωνε τα καλύτερα από τα στοιχεία των τριών ανεξάρτητων μέχρι τότε προσεγγίσεων. Υπάρχουν, ασφαλώς, και άλλες προσεγγίσεις αντικειμενοστρεφούς ανάπτυξης λογισμικού τις οποίες ο αναγνώστης ενθαρρύνεται να αναζητήσει στην προτεινόμενη βιβλιογραφία. Ωστόσο, δεν είναι περισσότερο διαδεδομένες από καμία εκ των τριών προσεγγίσεων που αποτέλεσαν το σπόρο για την ενοποιημένη προσέγγιση και με αυτό το σκεπτικό επιλέξαμε να αναφερθούμε σε αυτή, χωρίς αυτό να σημαίνει ότι είναι η μοναδική ή η καλύτερη.

Κατά την αναφορά στην ενοποιημένη προσέγγιση θα διατηρηθούν οι επιβεβλημένες αποστάσεις τόσο από τις κατασκευαστικές λεπτομέρειες, οι οποίες σχετίζονται με το περιβάλλον προγραμματισμού, όσο και από επιλογές συγκεκριμένων εταιρειών, έχοντας ως σκοπό να αποδώσουμε τα ουσιώδη ποιοτικά χαρακτηριστικά της αντικειμενοστρεφούς σκέψης.



## ΕΝΟΤΗΤΑ 8.1. ΕΝΑ ΓΕΝΙΚΟ ΠΛΑΙΣΙΟ ΓΙΑ ΤΗΝ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗ ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ

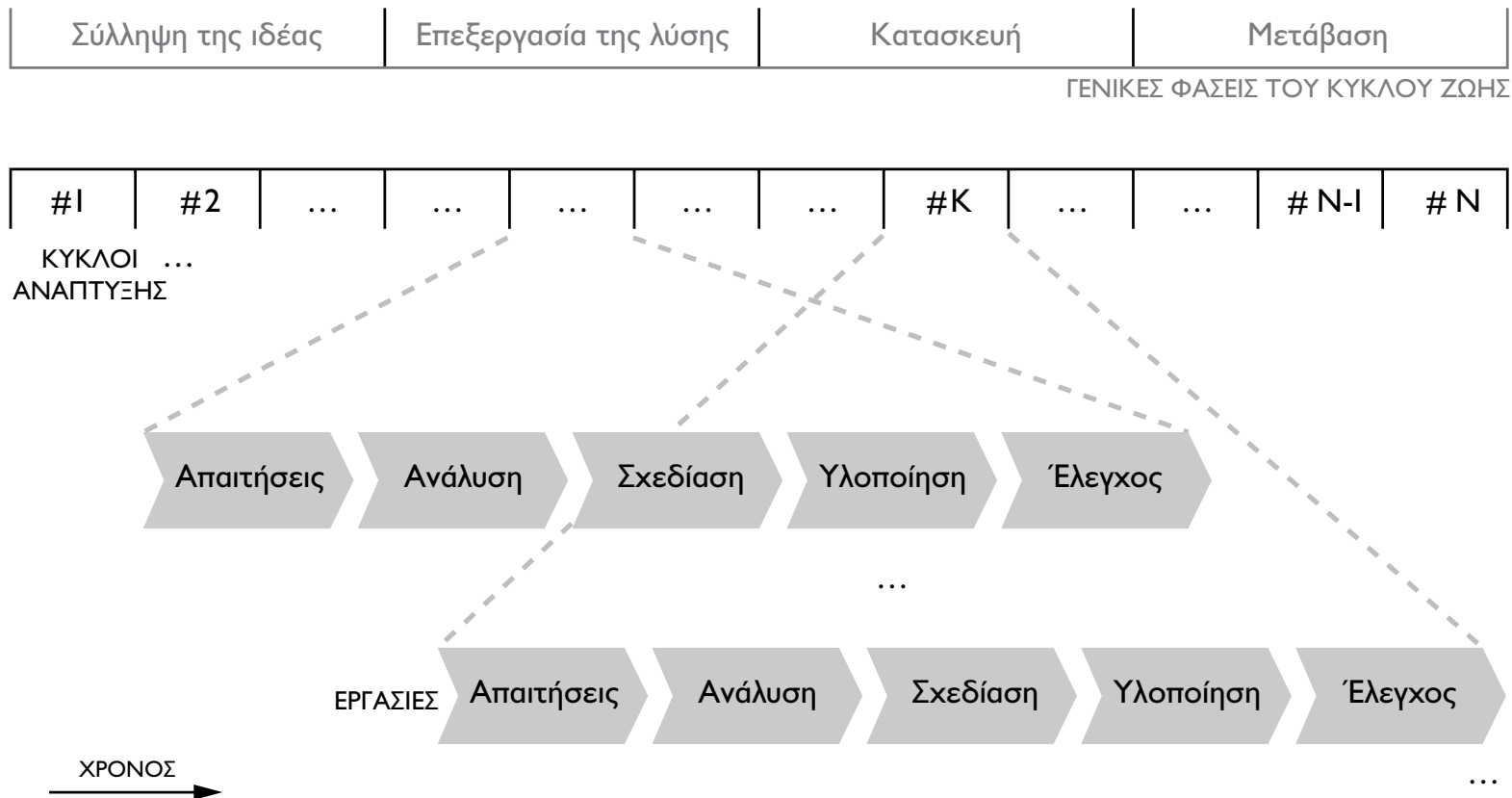
Η αυξανόμενη ζήτηση για ολοένα και πιο πολύπλοκες εφαρμογές λογισμικού έχει δημιουργήσει την απαίτηση ευέλικτων και αποτελεσματικών προσεγγίσεων στην ανάπτυξή του, όπως επισημάνσαμε στα δύο πρώτα κεφάλαια του βιβλίου αυτού. Μία από τις προσπάθειες να ικανοποιηθεί η απαίτηση αυτή είναι η ενοποιημένη προσέγγιση (Unified Software Development Methodology ή Unified Process), η οποία υποστηρίζει την ανάπτυξη λογισμικού σύμφωνα με την αντικειμενοστρεφή φιλοσοφία. Τα βασικά χαρακτηριστικά αυτής είναι τα ακόλουθα:

- Χρησιμοποιεί την UML για την παράσταση των μοντέλων λογισμικού που κατασκευάζονται κατά την ανάπτυξη.
- Αντιμετωπίζει το λογισμικό ως ένα σύνολο συστατικών που ικανοποιούν απαιτήσεις των χρηστών με αναφορά στις οποίες πραγματοποιεί όλες τις δραστηριότητες ανάπτυξης.
- Αντιμετωπίζει την αρχιτεκτονική του λογισμικού ως κεντρική έννοια στην ανάπτυξη, η οποία είναι δυναμικά αλληλεξαρτώμενη με τις απαιτήσεις των χρηστών, δηλαδή καθορίζεται από αυτές αλλά και τις επηρεάζει.
- Είναι μια επαναληπτική και επαυξητική προσέγγιση, δηλαδή χτίζει το τελικό προϊόν ως συσσωρευτικό αποτέλεσμα επαναλήψεων δραστηριοτήτων ανάπτυξης λογισμικού.

Η ενοποιημένη προσέγγιση ακολουθεί ένα γενικό μοντέλο κύκλου ζωής λογισμικού (βλ. Κεφάλαιο 2), όπως εξειδικεύεται στο Σχήμα 8.1 που ακολουθεί.



**Σχήμα 8.1** Το μοντέλο κύκλου ζωής της ενοποιημένης προσέγγισης ανάπτυξης λογισμικού.



Οι γενικές φάσεις που αναγνωρίζονται είναι αυτές της σύλληψης της ιδέας, της επεξεργασίας της λύσης, της κατασκευής και της μετάβασης. Κάθε γενική φάση αναλύεται σε κύκλους ανάπτυξης και σε κάθε κύκλο ανάπτυξης λαμβάνουν χώρα οι εργασίες της προδιαγραφής των απαιτήσεων, της ανάλυσης, της σχεδίασης, της υλοποίησης και του ελέγχου. Σε αυτό, όπως και στο επόμενο κεφάλαιο, θα ασχοληθούμε με την περιγραφή των εργασιών της προδιαγραφής των απαιτήσεων, της ανάλυσης και της σχεδίασης λογισμικού σύμφωνα με την ενοποιημένη προσέγγιση.

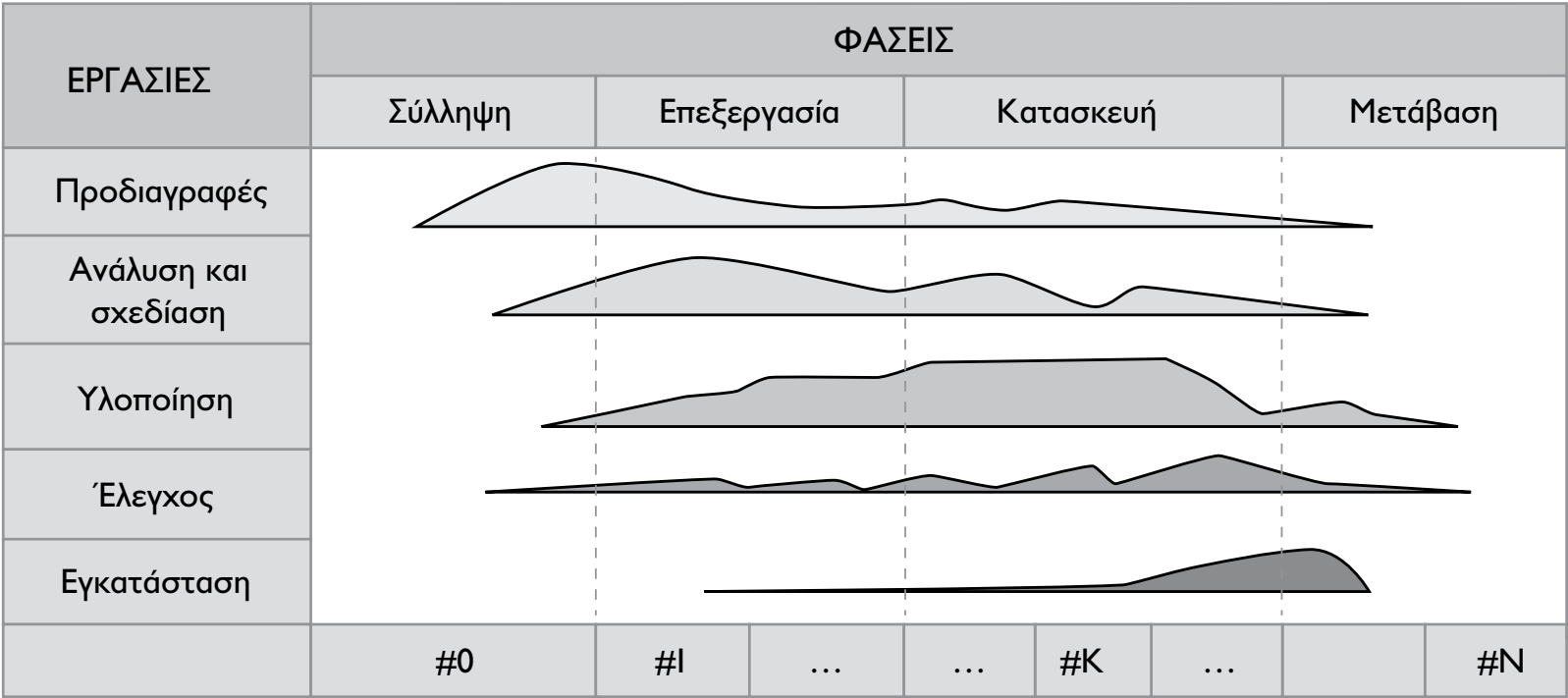
Κάθε κύκλος ανάπτυξης αφορά ένα υποσύνολο του λογισμικού υπό ανάπτυξη, το οποίο οριοθετείται είτε από το πλήθος των λειτουργιών που περιλαμβάνει είτε από το βαθμό λεπτομέρειας με τον οποίο τις αντιμετωπίζει. Στο τέλος κάθε κύκλου ανάπτυξης έχουμε μια εκδοχή (release) του λογισμικού. Κατά τις πρώτες δύο φάσεις (σύλληψη της ιδέας και επεξεργασία της λύσης) η εκδοχή αυτή είναι ένα σύνολο από κείμενα και μοντέλα παράστασης λογισμικού, ενώ κατά τη φάση της κατασκευής η εκδοχή αυτή περιλαμβάνει πηγαίο και εκτελέσιμο κώδικα με ολοένα και περισσότερα από τα απαιτούμενα λειτουργικά χαρακτηριστικά του λογισμικού. Τέλος, κατά τη φάση της μετάβασης το λογισμικό τοποθετείται σε δοκιμαστική λειτουργία, όπου επαληθεύεται η ικανοποίηση των απαιτήσεων των χρηστών και γίνονται οι απαραίτητες διορθώσεις.

Είναι πιθανόν αντιληπτό ότι δεν εκτελούνται όλες οι εργασίες (προδιαγραφή, ανάλυση, σχεδίαση, υλοποίηση, έλεγχος) σε όλες τις γενικές φάσεις. Για παράδειγμα, σύμφωνα με όσα γνωρίζουμε από τη δομημένη ανάλυση και σχεδίαση, δεν είναι δυνατό κατά τη φάση της σύλληψης της ιδέας να εκτελεστεί οποιαδήποτε εργασία σχεδίασης λογισμικού. Αυτό είναι εν μέρει μόνο αληθές στην περίπτωση της αντικειμενοστρεφούς τεχνολογίας, όπου θα δούμε ότι σχετικά νωρίς στην ανάπτυξη εισάγονται έννοιες όπως «κλάση» και «συνεργασία», οι οποίες τελικά καταλήγουν να είναι πραγματικά κλάσεις και στο πεδίο της υλοποίησης.

Ενδέχεται, ασφαλώς, οι οντότητες που εντοπίζονται στις πρώτες φάσεις ανάπτυξης του λογισμικού να μην απεικονιστούν στη συνέχεια αυτούσιες σε κλάσεις. Αυτό, ωστόσο, δεν αναιρεί το γεγονός ότι από πολύ νωρίς στην αντικειμενοστρεφή ανάπτυξη εισάγονται όροι συστατικών στοιχείων

υλοποίησης του λογισμικού. Έχοντας κατά νου τη σαφή διάκριση ανάλυσης και σχεδίασης η οποία ισχύει στη δομημένη προσέγγιση, παρατηρούμε ότι στην αντικειμενοστρεφή εκτελούμε νωρίς εργασίες που μπορούν να χαρακτηριστούν και ως εργασίες ανάλυσης και ως εργασίες σχεδίασης.

**Σχήμα 8.2** Μια απεικόνιση της αναλογίας των εργασιών ανάπτυξης λογισμικού κατά τις γενικές φάσεις της ενοποιημένης προσέγγισης.



Στο Σχήμα 8.2 φαίνεται η ποσότητα των πόρων που αποδίδονται στις εργασίες προδιαγραφής των απαιτήσεων, ανάλυσης και σχεδίασης, υλοποίησης, ελέγχου και εγκατάστασης κατά τους κύκλους ανάπτυξης της ενοποιημένης προσέγγισης ανάπτυξης λογισμικού. Το εμβαδόν μεταξύ του οριζόντιου άξονα του χρόνου και κάθε καμπύλης υποδηλώνει τους πόρους που ανατίθενται στην εκτέλεση κάθε εργασίας ή, ισοδύναμα, την ενέργεια που αποδίδεται σε αυτή.

Είναι ίσως αντιληπτό ότι το διάγραμμα δεν περιέχει ποσοτικές πληροφορίες αλλά σκοπεύει να μας δείξει ποιοτικά την αναλογία των εργασιών προδιαγραφής, ανάλυσης κ.λπ. σε κάθε κύκλο ανάπτυξης, καθώς και την παράλληλη εκτέλεση πολλών από αυτές. Παρατηρούμε ότι στις αρχικές φάσεις εκτελούνται περισσότερο εργασίες προδιαγραφής και λιγότερο ανάλυσης, σχεδίασης ή υλοποίησης. Όσο περνάμε σε κύκλους ανάπτυξης που ανήκουν στις επόμενες φάσεις, εκτελούνται περισσότερο κατασκευαστικές εργασίες ή έλεγχος και λιγότερο εργασίες προσδιορισμού των απαιτήσεων, οπότε μεταβάλλεται και το αντίστοιχο εμβαδόν.

Η ενοποιημένη προσέγγιση ανάπτυξης λογισμικού έχει το χαρακτηριστικό ότι, σε περίπτωση που κατά την ανάπτυξη εντοπιστεί κάποιο αδιέξοδο, το κόστος περιορίζεται σε αυτό της επανάληψης του τελευταίου κύκλου και όχι ολόκληρης της ανάπτυξης μέχρι το σημείο εκείνο. Βέβαια, η ιδέα δεν είναι νέα και χαρακτηρίζει και άλλα μοντέλα κύκλου ζωής λογισμικού, ανεξάρτητα από το αν αυτά ακολουθούν τη δομημένη ή την αντικειμενοστρεφή προσέγγιση.

## Σύνοψη ενότητας

---

*Η ενοποιημένη προσέγγιση αποτελεί αποτέλεσμα της σύγκλισης τριών από τις επικρατέστερες προσεγγίσεις, σύμφωνα με την αντικειμενοστρεφή φιλοσοφία ανάπτυξης λογισμικού. Χρησιμοποιεί την UML για την παράσταση όλων των συστατικών στοιχείων λογισμικού, θεωρώντας τα συστατικά αυτά ως την ικανοποίηση των απαιτήσεων των χρηστών του λογισμικού. Κεντρική έννοια στην ενοποιημένη προσέγγιση είναι η αρχιτεκτονική του λογισμικού, η οποία είναι συνυφασμένη με τις απαιτήσεις των χρηστών. Το μοντέλο κύκλου ζωής που ακολουθεί αποτελεί*

εξειδίκευση ενός γενικού μοντέλου κύκλου ζωής λογισμικού και χαρακτηρίζεται ως επαναληπτικό και επαυξητικό, δηλαδή το τελικό προϊόν κατασκευάζεται ως το συσσωρευτικό αποτέλεσμα επαναλήψεων των δραστηριοτήτων ανάπτυξης λογισμικού.

### **Δραστηριότητα I/Κεφάλαιο 8**

Προσπαθήστε να σχολιάσετε με 100-150 λέξεις το γεγονός ότι στην αντικειμενοστρεφή τεχνολογία από νωρίς στη διαδικασία ανάλυσης εισάγονται όροι υλοποίησης. Μπορείτε να κατευθύνετε τον προβληματισμό σας στο ερώτημα: «Πόσο ανεξάρτητη μπορεί να είναι η σκέψη του αναλυτή από τα εργαλεία που γνωρίζει ότι θα έχει στη διάθεσή του για τη σχεδίαση και την υλοποίηση;».

### **Άσκηση I/Κεφάλαιο 8**

Αρκετά από τα στοιχεία του μοντέλου κύκλου ζωής της ενοποιημένης προσέγγισης δεν είναι καινοτομίες που η ίδια εισήγαγε αλλά υπάρχουσες ιδέες που ενσωμάτωσε. Για να τεκμηριώσετε αυτήν τη θέση, ανατρέξτε στο δεύτερο κεφάλαιο του βιβλίου αυτού και εντοπίστε τα μοντέλα κύκλου ζωής λογισμικού που παρουσιάζουν το πλεονέκτημα του περιορισμού του κόστους σε περίπτωση εντοπισμού αδιεξόδου, στο κόστος της επανάληψης μόνο του τελευταίου κύκλου και όχι ολόκληρης της ανάπτυξης λογισμικού.

## ΕΝΟΤΗΤΑ 8.2. Η ΕΝΝΟΙΑ ΤΗΣ ΠΕΡΙΠΤΩΣΗΣ ΧΡΗΣΗΣ

Στην ενότητα αυτή θα εισάγουμε την έννοια της περίπτωσης χρήσης, θα δώσουμε τους συμβολισμούς που χρησιμοποιεί η UML για τις περιπτώσεις χρήσης και θα εντοπίσουμε τον κεντρικό ρόλο των περιπτώσεων χρήσης στην ενοποιημένη προσέγγιση ανάπτυξης λογισμικού.

### 8.2.1. Εισαγωγή

Όλα τα έργα ανάπτυξης λογισμικού ξεκινούν από τον ορισμό ενός προβλήματος του πραγματικού κόσμου. Ανεξάρτητα από τον τρόπο με τον οποίο ο ορισμός αυτός αναφέρεται στις διάφορες μεθοδολογίες, στα μοντέλα κύκλου ζωής και στα πρότυπα που χρησιμοποιούνται στην ανάπτυξη του λογισμικού, πρόκειται ουσιαστικά για το ίδιο πράγμα: την περιγραφή της ανάγκης των χρηστών για κατασκευή λογισμικού που θα πραγματοποιεί κάποιες επιθυμητές σε αυτούς λειτουργίες. Όπως αναφέραμε και στο πρώτο κεφάλαιο, η Τεχνολογία Λογισμικού πρέπει να υποστηρίζει τους κατασκευαστές στην παραγωγή λογισμικού που ικανοποιεί τις απαιτήσεις των χρηστών.

Αφήνοντας –προς το παρόν– κατά μέρους και άλλες απαιτήσεις από μια αποτελεσματική προσέγγιση στην ανάπτυξη λογισμικού, όπως η τεχνική ορθότητα, η αποφυγή σφαλμάτων και η κατασκευή εντός χρονοδιαγράμματος και προϋπολογισμού, θα σχολιάσουμε για λίγο το θέμα της ικανοποίησης των απαιτήσεων των χρηστών. Υπενθυμίζουμε δύο ουσιώδη χαρακτηριστικά της δομημένης ανάλυσης και σχεδίασης:

- Δεν είναι δυνατόν να προχωρήσουμε στον ορισμό της δομής προγράμματος, ακόμη και για ένα μικρό τμήμα της εφαρμογής λογισμικού, αν δεν έχουμε πλήρη ορισμό των λειτουργικών απαιτήσεων και το διάγραμμα ροής δεδομένων που αφορούν το τμήμα αυτό της εφαρμογής. Οποιαδήποτε τροποποίηση των απαιτήσεων των χρηστών είναι επίπονη και συνήθως επιφέρει αναταράξεις.
- Η τελευταία επαφή του κατασκευαστή με τις απαιτήσεις των χρηστών είναι η αποτύπωσή τους στο έγγραφο προδιαγραφών των απαιτήσεων από το λογισμικό και στο διάγραμμα ροής δεδομένων.



Από το σημείο εκείνο και μετά οι απαιτήσεις «κλειδώνουν» και η συνέχεια της ανάπτυξης γίνεται χωρίς τα συστατικά λογισμικού που παράγονται να έχουν εμφανή σύνδεση με τις συγκεκριμένες απαιτήσεις στην ικανοποίηση των οποίων στοχεύουν.

Κανένα από τα δύο αυτά σημεία δεν χαρακτηρίζει την αντικειμενοστρεφή ανάλυση και σχεδίαση με την ενοποιημένη προσέγγιση. Όπως φαίνεται και από το μοντέλο κύκλου ζωής που απεικονίζεται στο Σχήμα 8.1 αλλά και από το Σχήμα 8.2, οι απαιτήσεις απασχολούν τον κατασκευαστή σε όλη τη διάρκεια της ανάπτυξης και αποτελούν σημείο αναφοράς όλων των συστατικών λογισμικού που παράγονται κατά την ανάπτυξη. Επίσης, όπως θα αναφέρουμε στη συνέχεια, η αρχιτεκτονική στην αντικειμενοστρεφή ανάπτυξη λογισμικού ορίζεται από νωρίς και δύναται να ωριμάζει μαζί με τις απαιτήσεις.

Αυτό, ασφαλώς, δεν σημαίνει ότι η αντικειμενοστρεφής ανάπτυξη λογισμικού είναι γενικά δεκτική σε μεταβολές των απαιτήσεων όποτε και αν αυτές συμβαίνουν. Πάντα οι μεταβολές στις απαιτήσεις δημιουργούν προβλήματα και επιφέρουν κόστος στην ανάπτυξη λογισμικού, η οποία δεν θα τελείωνε ποτέ αν περιμέναμε να ωριμάσουν πλήρως οι απαιτήσεις των χρηστών. Η αντικειμενοστρεφής φιλοσοφία είναι λίγο πιο ελαστική απ' ό,τι η δομημένη σε τέτοιες μεταβολές, οι οποίες είναι αποδεκτό να συμβαίνουν σε μικρή έκταση κατά την ανάπτυξη λογισμικού.

### **8.2.2. Τι είναι «περίπτωση χρήσης»;**

Η ικανοποίηση κάθε λειτουργικής απαίτησης από μία εφαρμογή λογισμικού υλοποιείται ως μια αλληλουχία ενεργειών που εκτελούνται από το λογισμικό, αλληλεπιδρώντας είτε με κάποιον χρήστη (φυσικό πρόσωπο) είτε με άλλα συστήματα (λ.χ. άλλες εφαρμογές λογισμικού, εξωτερικές συσκευές, εξωτερικές πηγές δεδομένων). Μια τέτοια αλληλεπίδραση παράγει ένα αποτέλεσμα επιθυμητό για το χρήστη της εφαρμογής λογισμικού, δηλαδή ικανοποιεί μια λειτουργική απαίτησή του και ονομάζεται «περίπτωση χρήσης».



### Περίπτωση χρήσης:

Μια περίπτωση χρήσης (use case) είναι μια αλληλουχία ενεργειών που εκτελεί το λογισμικό αλληλεπιδρώντας με τον χρήστη ή με εξωτερικά συστήματα, προκειμένου να ικανοποιήσει μία λειτουργική απαίτηση.

Κάθε περίπτωση χρήσης μπορεί να περιγράφεται με μεγαλύτερη ή μικρότερη λεπτομέρεια, όπως άλλωστε και κάθε απαίτηση από το λογισμικό. Για παράδειγμα, μπορούμε να ορίσουμε μια περίπτωση χρήσης γενικά ως «τήρηση αρχείου σπουδαστών» αλλά και με μεγαλύτερη λεπτομέρεια ως «εισαγωγή νέου σπουδαστή», «μεταβολή στοιχείων σπουδαστή» και «διαγραφή σπουδαστή». Στο σημείο αυτό δεν έχει μεγάλη σημασία το επίπεδο λεπτομέρειας, όσο ένα άλλο χαρακτηριστικό του ορισμού της περίπτωσης χρήσης το οποίο μπορεί εύκολα να περάσει απαρατήρητο:

Μια περίπτωση χρήσης χαρακτηρίζεται τόσο από την αλληλουχία των ενεργειών που εκτελεί το λογισμικό όσο και από το μέρος εκείνο με το οποίο αλληλεπιδρά, δηλαδή ένα ρόλο χρήστη (φυσικό πρόσωπο) ή ένα εξωτερικό σύστημα. Το μέρος αυτό ονομάζεται «χειριστής».

### Χειριστής:

Ένας χειριστής (actor) είναι μια κατηγορία χρηστών ή μια εξωτερική οντότητα με την οποία αλληλεπιδρά το λογισμικό κατά την εκτέλεση των ενεργειών μιας περίπτωσης χρήσης.

Στην περίπτωση που ένας χειριστής αντιστοιχεί σε χρήστη – φυσικό πρόσωπο κάνουμε λόγο για μια κατηγορία φυσικών προσώπων και όχι για κάποιο συγκεκριμένο πρόσωπο. Αυτό συμβαίνει διότι μας απασχολεί ο καθορισμός της αλληλεπίδρασης του ρόλου ενός χρήστη με το λογισμικό και ασφαλώς όχι η ταυτότητά του ιδίου του χρήστη. Αυτή η αλληλεπίδραση μπορεί να ιδωθεί ως ο ρόλος που παίζει ένα φυσικό πρόσωπο όταν χρησιμοποιεί το λογισμικό.

## Ρόλος:

Όταν ένας χειριστής αντιστοιχεί σε κατηγορία χρηστών λογισμικού – φυσικών προσώπων, τότε η έννοια του χειριστή είναι ισοδύναμη με την έννοια ενός ρόλου (role) των χρηστών του λογισμικού.

Στην περίπτωση που ο χειριστής αντιστοιχεί σε εξωτερικό σύστημα (λογισμικό, συσκευή), τότε συνήθως το σύστημα αυτό είναι κατά κανόνα συγκεκριμένο και πρέπει σε επόμενη φάση της ανάπτυξης να προδιαγραφεί πλήρως η διεπαφή (interface) του λογισμικού με αυτό. Εξαίρεση σε αυτό αποτελεί η περίπτωση όπου το λογισμικό που κατασκευάζεται προορίζεται να παρέχει υπηρεσίες σε άλλα συστήματα λογισμικού, οπότε ο χειριστής δεν αντιστοιχεί σε γνωστό εκ των προτέρων εξωτερικό σύστημα. Λαμβάνοντας υπόψη τις δύο τελευταίες παρατηρήσεις, μπορούμε να διατυπώσουμε τη θέση ότι στην ενοποιημένη προσέγγιση:

Το σύνολο των χειριστών μιας εφαρμογής λογισμικού αποτελεί το περιβάλλον λειτουργίας της.

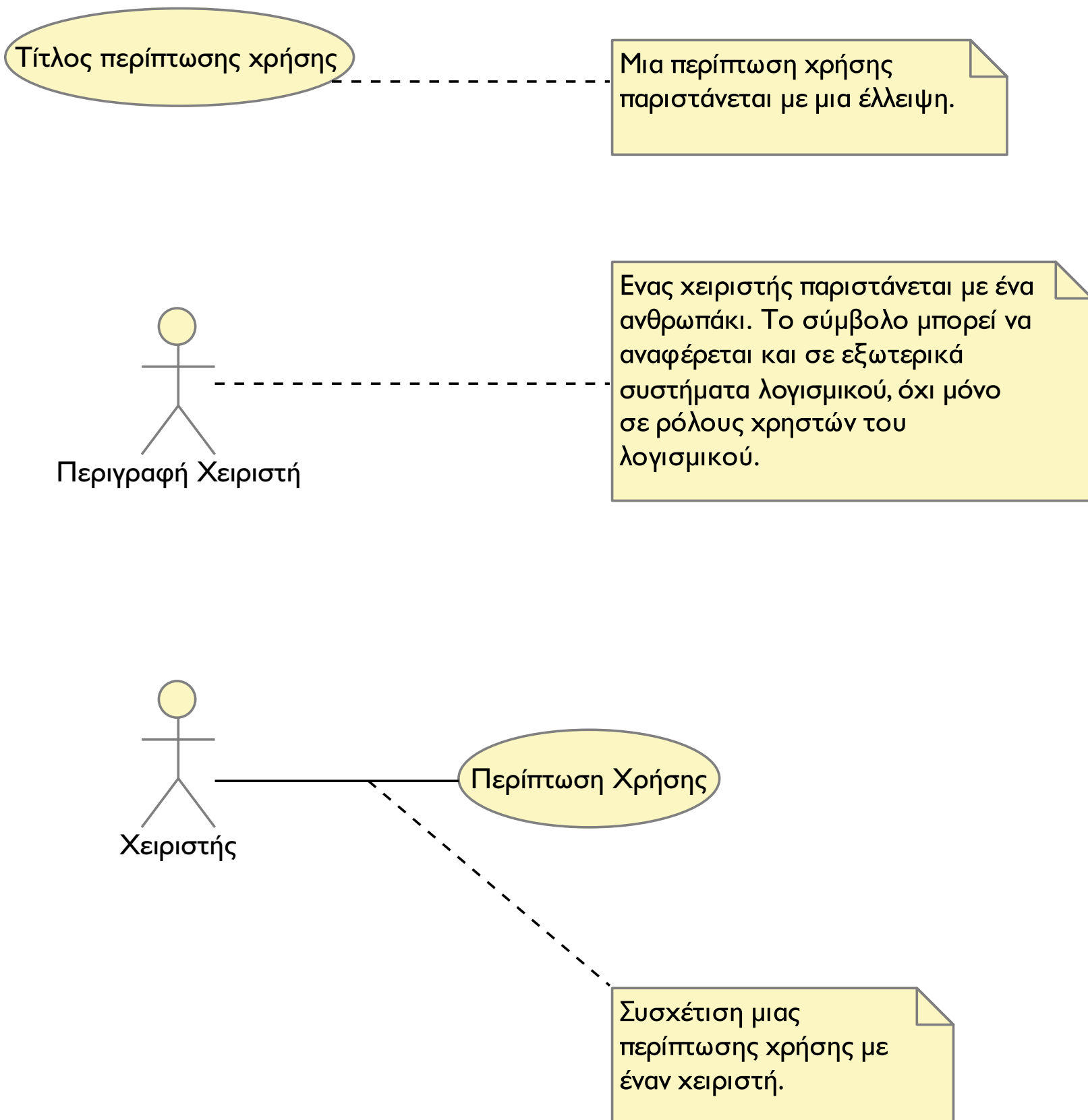
Κάθε περίπτωση χρήσης ενεργοποιείται από έναν χειριστή. Όταν εκτελούνται οι ενέργειες που περιλαμβάνονται στην περίπτωση χρήσης, τότε μπορούμε να λέμε ότι εκτελείται η περίπτωση χρήσης.

Η σαφής διάκριση των δύο αλληλεπιδρώντων μερών (περίπτωσης χρήσης και χειριστής) εμπεριέχει μια ουσιώδη διαφορά της ενοποιημένης προσέγγισης ανάπτυξης λογισμικού από άλλες προσεγγίσεις στον τομέα του καθορισμού των λειτουργικών απαιτήσεων από το λογισμικό. Συνήθως ο καθορισμός των λειτουργικών απαιτήσεων γίνεται απαντώντας στο ερώτημα: «Τι πρέπει να κάνει το λογισμικό;». Στην ενοποιημένη προσέγγιση, το ερώτημα εξειδικεύεται στο «Τι πρέπει να κάνει το λογισμικό για κάθε χειριστή αυτού;».

Μολονότι εκ πρώτης όψεως τα δύο ερωτήματα φαίνονται ισοδύναμα, η εισαγωγή της έννοιας του χειριστή μάς οδηγεί να σκεπτόμαστε με όρους αποτελεσματικότητας για τους χρήστες και το περιβάλλον της εφαρμογής.

Αυτό σημαίνει ότι αναζητούμε τις λειτουργικές απαιτήσεις ως εργασίες συνυφασμένες με τους ωφελούμενους από αυτές.

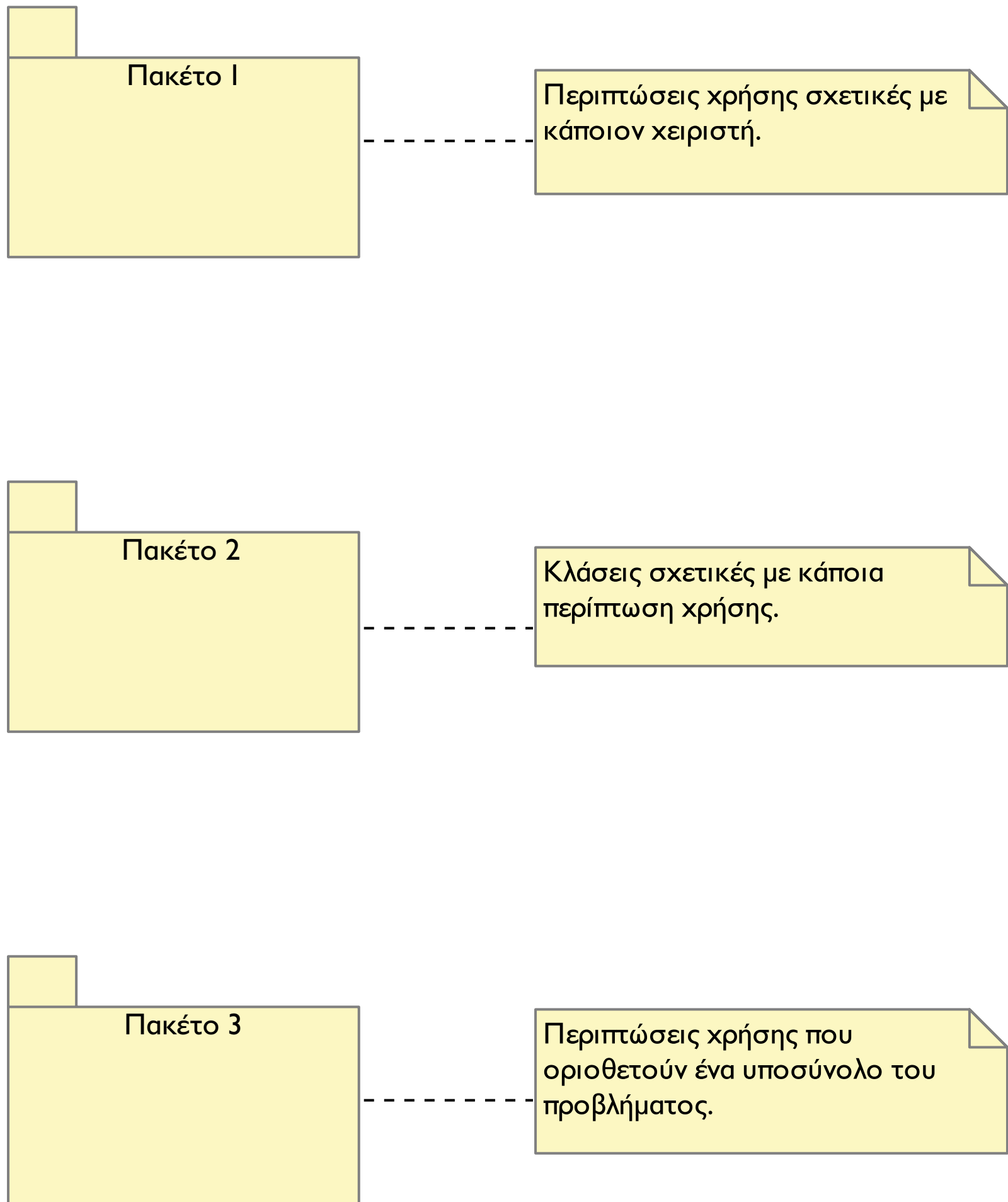
**Σχήμα 8.3** Συμβολισμοί UML για τις περιπτώσεις χρήσης και τους χειριστές.



Στο Σχήμα 8.3 φαίνονται οι συμβολισμοί που χρησιμοποιεί η UML για την παράσταση των περιπτώσεων χρήσης. Μια απεικόνιση περιπτώσεων χρήσης με τους συμβολισμούς αυτούς αποτελεί ένα διάγραμμα περιπτώσεων χρήσης (Use Case Diagram). Το σύνολο των διαγραμμάτων περιπτώσεων χρήσης αποτελεί το μοντέλο περιπτώσεων χρήσης της εφαρμογής (Use Case Model), το οποίο είναι ένα μοντέλο παράστασης λογισμικού.

Οι λειτουργικές απαιτήσεις από μια εφαρμογή λογισμικού αποτελούν, σύμφωνα με την ενοποιημένη προσέγγιση, περιπτώσεις χρήσης. Είναι κατανοητό ότι σε μια πραγματική εφαρμογή λογισμικού το πλήθος των περιπτώσεων χρήσης μπορεί να είναι πολύ μεγάλο για να μπορεί να απεικονιστεί με τη βοήθεια ενός και μόνο διαγράμματος το οποίο να είναι πρακτικό και αναγνώσιμο. Στη γενική περίπτωση ένα μοντέλο περιπτώσεων χρήσης αποτελείται από πολλά διαγράμματα, τα οποία μπορούν να εκτείνονται σε βάθος και να ομαδοποιούνται σε πακέτα (packages) συναφών για τον κατασκευαστή ή για το πρόβλημα περιπτώσεων χρήσης. Τα πακέτα είναι ένας πολύ χρήσιμος μηχανισμός ομαδοποίησης συστατικών και διαγραμμάτων στη UML, ο δε συμβολισμός τους δίνεται ευθύς αμέσως.

**Σχήμα 8.4** Τα πακέτα (packages) ως μηχανισμός ομαδοποίησης συστατικών λογισμικού στη UML.



Στο Σχήμα 8.4 φαίνεται ο συμβολισμός της UML για τα πακέτα. Κάθε πακέτο μπορεί να περιέχει οποιοδήποτε σύνολο από διαγράμματα, κλάσεις, περιπτώσεις χρήσης και άλλα συστατικά λογισμικού, καθώς και άλλα πακέτα.

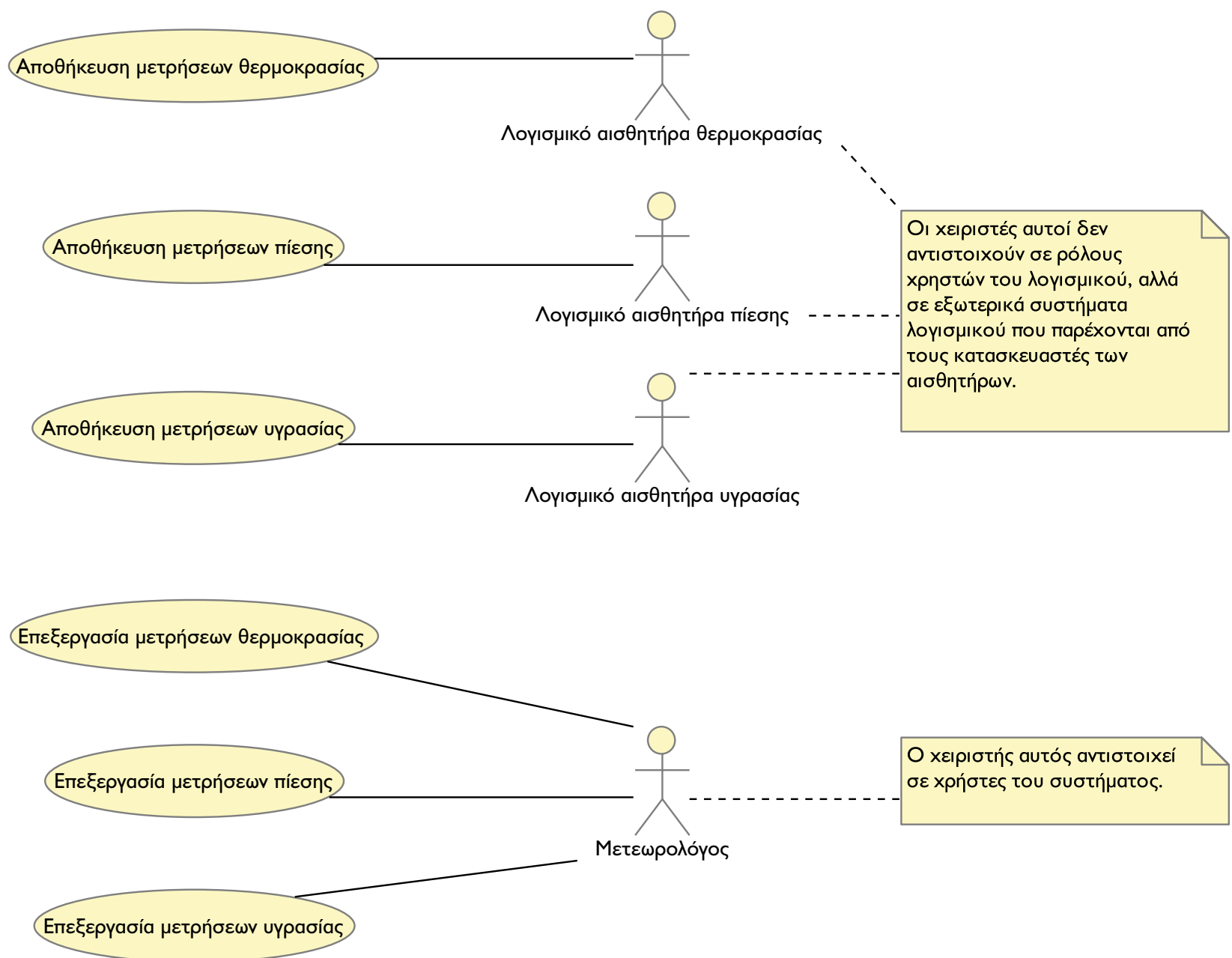
### Παράδειγμα I/Κεφάλαιο 8

**Ορισμός του προβλήματος:** Ζητείται η κατασκευή ενός συστήματος παρακολούθησης μετεωρολογικών μετρήσεων το οποίο με χρήση ειδικών αισθητήριων οργάνων συλλέγει από διάφορα γεωγραφικά σημεία δεδομένα θερμοκρασίας, ατμοσφαιρικής πίεσης και υγρασίας. Το σύστημα αποθηκεύει τα στοιχεία αυτά και, κατόπιν, εξάγει στατιστικά αποτελέσματα, όπως μέση τιμή και τυπική απόκλιση για κάθε γεωγραφικό σημείο. Το σύστημα αποτελείται από συσκευές μέτρησης (αισθητήρες) πίεσης, θερμοκρασίας και υγρασίας, από ηλεκτρονικούς υπολογιστές και από ανθρώπους.

**Λειτουργικές απαιτήσεις:** 1) Αποθήκευση μετρήσεων θερμοκρασίας. 2) Αποθήκευση μετρήσεων πίεσης. 3) Αποθήκευση μετρήσεων υγρασίας. 4) Επεξεργασία μετρήσεων θερμοκρασίας. 5) Επεξεργασία μετρήσεων πίεσης. 6) Επεξεργασία μετρήσεων υγρασίας.

Θα συμπληρώσουμε τον σύντομο ορισμό του προβλήματος με τη θεώρηση ότι καθένας από τους αισθητήρες θερμοκρασίας, ατμοσφαιρικής πίεσης και υγρασίας δίνει τις μετρήσεις του αυτόματα, μέσω ειδικού λογισμικού (driver) που τον συνδέει στον υπολογιστή στον οποίο τρέχει η εφαρμογή λογισμικού που θέλουμε να κατασκευάσουμε. Η περιγραφή των λειτουργικών απαιτήσεων ως περιπτώσεις χρήσης φαίνεται στο Σχήμα 8.5.

**Σχήμα 8.5** Το διάγραμμα περιπτώσεων χρήσης του παραδείγματος 8.1.





Είναι σκόπιμο να αναφέρουμε ότι το διάγραμμα περιπτώσεων χρήσης δεν είναι αντίστοιχο με το διάγραμμα ροής δεδομένων στη δομημένη ανάλυση, δηλαδή δεν περιγράφει ροές δεδομένων, γεγονότων ή ελέγχου. Περιγράφει απλώς τη συσχέτιση των χειριστών με περιπτώσεις χρήσης, δηλαδή με λειτουργικές απαιτήσεις. Το βέλος που συνδέει κάθε περίπτωση χρήσης με έναν χειριστή υποδηλώνει το γεγονός ότι η έναρξη των λειτουργιών που περιλαμβάνει μια περίπτωση χρήσης προκαλείται από κάποιον χειριστή. Η αναλυτική περιγραφή της αλληλουχίας των λειτουργιών αυτών γίνεται κατά την ανάλυση των περιπτώσεων χρήσης, όπως θα δούμε στη συνέχεια.

### **Δραστηριότητα 2/Κεφάλαιο 8**

Προσπαθήστε να δημιουργήσετε και μόνοι σας το διάγραμμα περιπτώσεων χρήσης που δίνεται στο Σχήμα 8.5 χρησιμοποιώντας ένα εργαλείο όπως το Visual Paradigm, το οποίο διαθέτει δωρεάν έκδοση που μπορείτε να κατεβάσετε από το διαδίκτυο.

### **Δραστηριότητα 3/Κεφάλαιο 8**

Θα συμπληρώσουμε τις λειτουργικές απαιτήσεις του λογισμικού συλλογής και επεξεργασίας μετεωρολογικών μετρήσεων του Παραδείγματος Ι με τις ακόλουθες απαιτήσεις: 7) Ρύθμιση ευαισθησίας (calibration) αισθητήρων και 8) Λήψη αντιγράφων ασφαλείας (backup) μετρήσεων.

Προσπαθήστε να συμπληρώσετε το διάγραμμα περιπτώσεων χρήσης που δίνεται στο Σχήμα 8.5 έχοντας υπόψη πως οι μετεωρολόγοι δεν υποχρεούνται να γνωρίζουν τεχνικά ζητήματα αισθητήρων ούτε και χειρισμό συσκευών λήψης αντιγράφων ασφαλείας.

### 8.2.3. Πώς προδιαγράφεται μια περίπτωση χρήσης;

Μέχρι το σημείο αυτό είδαμε ότι μια περίπτωση χρήσης χαρακτηρίζεται από έναν σύντομο τίτλο. Έχοντας κατά νου ότι οι περιπτώσεις χρήσης αντιστοιχούν στις λειτουργικές απαιτήσεις οι οποίες μας απασχόλησαν στο Κεφάλαιο 4, αναμένουμε ότι η πλήρης προδιαγραφή των περιπτώσεων χρήσης είναι αντίστοιχη με αυτή των λειτουργικών απαιτήσεων. Πράγματι, στο Σχήμα 8.6 φαίνεται η δομή της προδιαγραφής μιας περίπτωσης χρήσης σύμφωνα με την ενοποιημένη προσέγγιση. Με πλάγιους χαρακτήρες δίνεται μια σύντομη περιγραφή των περιεχομένων κάθε ενότητας της προδιαγραφής.

## Σχήμα 8.6 Μια δομή κειμένου για την πλήρη περιγραφή μιας περίπτωσης χρήσης.

### ΠΡΟΔΙΑΓΡΑΦΗ ΠΕΡΙΠΤΩΣΗΣ ΧΡΗΣΗΣ

#### 1. Τίτλος περίπτωσης χρήσης

Αναγράφεται ο τίτλος της περίπτωσης χρήσης.

#### 2. Σύντομη περιγραφή

Δίνεται μια πολύ σύντομη περιγραφή της περίπτωσης χρήσης σε 2-3 προτάσεις.

#### 3. Ροή γεγονότων

##### 3.1 Βασική ροή

Κάθε περίπτωση χρήσης ξεκινά με μια ενέργεια ενός χειριστή. Στην παράγραφο αυτή περιγράφεται τι κάνει ο χειριστής και ποια είναι η ακολουθία των ενεργειών του λογισμικού με τις οποίες επιτυγχάνεται ο σκοπός της περίπτωσης χρήσης, χωρίς να περιγράφεται το γιατί ή το πώς πραγματοποιούνται οι ενέργειες αυτές. Μπορούν να χρησιμοποιηθούν σχήματα ή εικόνες που συμβάλλουν στην κατανόηση της ροής.

##### 3.2 Εναλλακτικές ροές

###### 3.2.1 Εναλλακτική ροή 1

###### 3.2.2 Εναλλακτική ροή 2

...

Εδώ περιγράφονται τυχόν εναλλακτικές ροές ενεργειών που μπορεί να συμβούν κατά την υλοποίηση της περίπτωσης χρήσης, όπως η απαιτούμενη συμπεριφορά του λογισμικού σε περίπτωση κάποιου σφάλματος.

#### 4. Μη λειτουργικές απαιτήσεις

##### 4.1 Απαίτηση 1

##### 4.2 Απαίτηση 2

...

Εδώ περιγράφονται οι μη λειτουργικές απαιτήσεις που σχετίζονται με την περίπτωση χρήσης, όπως απαιτήσεις επίδοσης ή περιβάλλοντος.

#### 5. Κατάσταση εισόδου

##### 5.1 Συνθήκη εισόδου 1

##### 5.2 Συνθήκη εισόδου 2

...

Στην ενότητα αυτή αναφέρονται οι συνθήκες που θα πρέπει να ισχύουν (προαπαιτήσεις, pre-conditions) για την εφαρμογή της περίπτωσης χρήσης, όπως τα αναγκαία δικαιώματα χρήστη ή οι συσκευές που πρέπει να είναι διαθέσιμες.

#### 6. Κατάσταση εξόδου

##### 6.1 Συνθήκη εξόδου 1

##### 6.2 Συνθήκη εξόδου 2

...

Στην ενότητα αυτή αναφέρονται οι συνθήκες που ισχύουν (αποτελέσματα, post-conditions) μετά την εφαρμογή της περίπτωσης χρήσης, όπως μεταβολές στην κατάσταση πόρων του συστήματος, συσκευών κ.ά.

Η προδιαγραφή μιας περίπτωσης χρήσης πρέπει να καθορίζει την κύρια ακολουθία (και ενδεχομένως και τις εναλλακτικές, εφόσον υπάρχουν) των λειτουργιών του συστήματος οι οποίες αποτελούν την περίπτωση χρήσης, χωρίς να περιλαμβάνει καθόλου κατασκευαστικές λεπτομέρειες για τον τρόπο πραγματοποίησης των λειτουργιών αυτών. Το μοντέλο περιπτώσεων χρήσης μαζί με το σύνολο των προδιαγραφών αυτών πρέπει να είναι κατανοητά τόσο από τον κατασκευαστή όσο και από τον πελάτη. Μερικές χρήσιμες παρατηρήσεις σχετικά με τον ορισμό και την προδιαγραφή των περιπτώσεων χρήσης είναι οι ακόλουθες.

- Μια περίπτωση χρήσης οριοθετείται από την ικανοποίηση μιας απαίτησης ενός χειριστή. Δεν αφορά μερική ικανοποίηση της απαίτησης ούτε ικανοποίηση ενός συνόλου απαιτήσεων του χειριστή και δεν ορίζεται έχοντας κατά νου τα συστατικά στοιχεία λογισμικού που θα την υλοποιήσουν.
- Δεν είναι δυνατό να εντοπιστούν από την αρχή με τρόπο αδιαφιλονίκητο όλες οι περιπτώσεις χρήσης. Κάτι τέτοιο θα ήταν ισοδύναμο με την απαίτηση της δομημένης φιλοσοφίας να είναι πλήρως γνωστές οι προδιαγραφές του λογισμικού πριν ξεκινήσει η ανάλυση και η σχεδίαση. Οι περιπτώσεις χρήσης πρέπει να ωριμάζουν καθώς η ανάπτυξη προχωρεί.
- Δεν είναι εύκολη η πλήρης προδιαγραφή μιας περίπτωσης χρήσης με την πρώτη προσπάθεια. Η λεπτομέρεια και η ακρίβεια της προδιαγραφής θα πρέπει να ακολουθεί την ωρίμανση των περιπτώσεων χρήσης και να αυξάνει σε κάθε κύκλο ανάπτυξης.
- Αν για το σαφή προσδιορισμό της περίπτωσης χρήσης απαιτούνται διαγράμματα και εικόνες όπως, λόγου χάρη, για την περιγραφή της διεπαφής με τον χρήστη ή των εναλλακτικών ροών, αυτά θα πρέπει να χρησιμοποιούνται αφειδώς και να ακολουθούν συμβολισμούς που ορίζει η UML. Ένα τέτοιο διάγραμμα είναι το διάγραμμα δραστηριοτήτων που θα εισάγουμε στη συνέχεια.
- Τονίζεται ότι κατά την προδιαγραφή των περιπτώσεων χρήσης γίνεται λόγος μόνο για τις ενέργειες που θα πραγματοποιούνται από το λογισμικό. Όχι για το πώς ούτε για το γιατί αυτές θα πραγματοποιούνται.

- Υπάρχουν αρκετές περιπτώσεις όπου η περιγραφή των περιπτώσεων χρήσης με τη βοήθεια της δομής που δόθηκε είναι πλεονασμός. Αν μια περίπτωση χρήσης είναι πολύ απλή, τότε μπορεί να περιγράφεται από τις δύο πρώτες ενότητες που αναφέρονται στο Σχήμα 8.6. Χρειάζεται, όμως, προσοχή στο χαρακτηρισμό μιας περίπτωσης χρήσης ως «απλής». Ένα κριτήριο είναι η κοινή αντίληψη της σύντομης περιγραφής της περίπτωσης χρήσης τόσο από τον πελάτη όσο και από τον κατασκευαστή.

Το έγγραφο προδιαγραφών των απαιτήσεων από το λογισμικό αναδομείται προκειμένου να ενσωματώσει την έννοια της περίπτωσης χρήσης. Μια νέα προτεινόμενη δομή είναι αυτή που δίνεται στο Σχήμα 8.7.

## **Σχήμα 8.7** Μια δομή του εγγράφου προδιαγραφών των απαιτήσεων από το λογισμικό για την ενοποιημένη προσέγγιση.

### ΕΓΓΡΑΦΟ ΠΡΟΔΙΑΓΡΑΦΩΝ ΤΩΝ ΑΠΑΙΤΗΣΕΩΝ ΑΠΟ ΤΟ ΛΟΓΙΣΜΙΚΟ

#### **I. Εισαγωγή**

- I.1 Σκοπός
- I.2 Εμβέλεια
- I.3 Ορισμοί, ακρωνύμια, συντομογραφίες
- I.4 Αναφορές
- I.5 Γενική εικόνα

#### **2. Το μοντέλο περιπτώσεων χρήσης**

- 2.1 Στίγμα
- 2.2 Προοπτική
- 2.3 Περιορισμοί
- 2.4 Γενική εικόνα
- 2.5 Παραδοχές και εξαρτήσεις

#### **3. Ανάλυση περιπτώσεων χρήσης**

- 3.1 Προδιαγραφές περιπτώσεων χρήσης
  - 3.1.1 Περίπτωση χρήσης 1
  - 3.1.2 Περίπτωση χρήσης 2
  - ...

*Εδώ τοποθετούνται όλες οι προδιαγραφές των περιπτώσεων χρήσης, σύμφωνα με τη δομή που δόθηκε στο Σχήμα 8.6.*

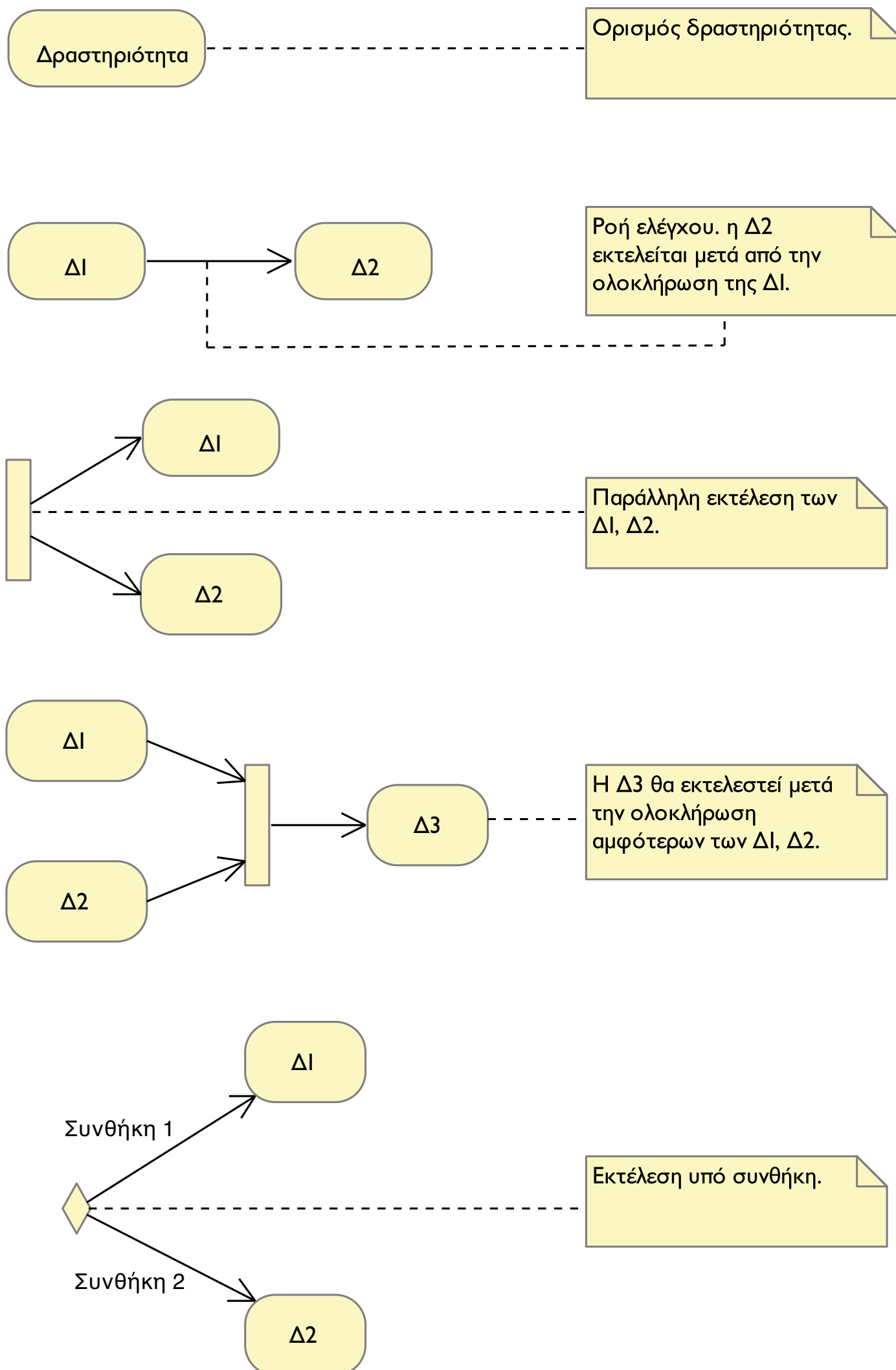
#### **4. Συμπληρωματικά στοιχεία**

- 4.1 Περιγραφή διαπροσωπειών
- 4.2 Παραρτήματα

## Συμβολισμοί UML

Ένα χρήσιμο εργαλείο για την περιγραφή της ροής των εργασιών σε μια περίπτωση χρήσης είναι το διάγραμμα δραστηριότητας (activity diagram) της UML. Το διάγραμμα αυτό μπορεί να χρησιμοποιηθεί συμπληρωματικά με την περιγραφή κειμένου για να προδιαγράψει μια περίπτωση χρήσης. Στη συνέχεια θα το χρησιμοποιούμε και για την περιγραφή της ροής εργασιών ανάπτυξης λογισμικού. Οι συμβολισμοί του διαγράμματος δραστηριότητας δίνονται στο Σχήμα 8.8 και εισάγονται –ασφαλώς– με χρήση UML.

**Σχήμα 8.8** Συμβολισμοί UML για το διάγραμμα δραστηριοτήτων (activity diagram).



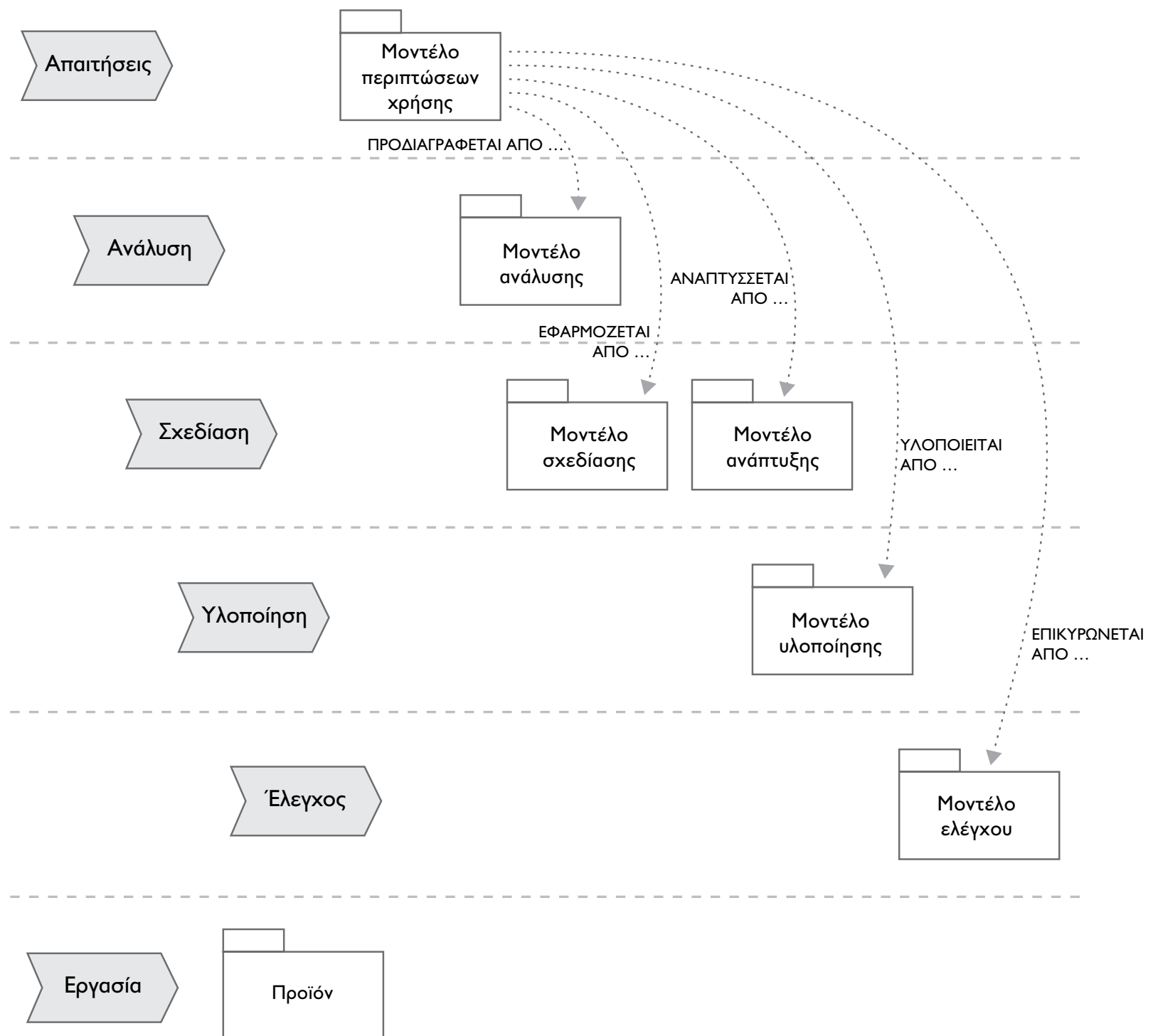


#### 8.2.4. Ένα σημείο αναφοράς

Οι περιπτώσεις χρήσης δεν είναι απλά ένας άλλος τρόπος περιγραφής των απαιτήσεων από το λογισμικό. Στην ενοποιημένη προσέγγιση, οι περιπτώσεις χρήσης παίζουν έναν κεντρικό ρόλο και λειτουργούν ως σημείο αναφοράς για ολόκληρη την ανάπτυξη. Αυτό σημαίνει πως είναι δυνατός ο εντοπισμός της περίπτωσης χρήσης με την ικανοποίηση της οποίας σχετίζεται ένα συστατικό λογισμικού σε οποιαδήποτε φάση του κύκλου ζωής και αν αυτό παράγεται.

Στο Σχήμα 8.9 φαίνεται με ποιο τρόπο το μοντέλο περιπτώσεων χρήσης το οποίο παράγεται ως προϊόν της εργασίας προσδιορισμού των απαιτήσεων από το λογισμικό σχετίζεται με όλα τα προϊόντα των εργασιών που έπονται.

**Σχήμα 8.9** Ο κεντρικός ρόλος των περιπτώσεων χρήσης στην ενοποιημένη προσέγγιση.



Το μοντέλο ανάλυσης, το οποίο θα δούμε αναλυτικά πιο κάτω, είναι μια λεπτομερής προδιαγραφή των περιπτώσεων χρήσης και αποτελεί την αρχική εκδοχή του μοντέλου σχεδίασης. Περιέχει την πρώτη ποιοτική εκδοχή της υλοποίησης των περιπτώσεων χρήσης σε όρους συστατικών στοιχείων λογισμικού. Το μοντέλο σχεδίασης εξειδικεύει το μοντέλο ανάλυσης και περιέχει καλά προσδιορισμένες κλάσεις και σχέσεις μεταξύ αυτών, οι οποίες εφαρμόζουν τις απαιτήσεις που περιγράφουν οι περιπτώσεις χρήσης.

Το μοντέλο διάταξης (deployment model) περιγράφει τη φυσική κατανομή των συστατικών του λογισμικού σε υπολογιστικούς πόρους, δηλαδή τη διάταξη του λογισμικού, όπως αυτή ορίστηκε στο Κεφάλαιο 3. Το μοντέλο υλοποίησης είναι το σύνολο του πηγαίου κώδικα, δηλαδή αυτή καθεαυτή η υλοποίηση του μοντέλου σχεδίασης και, κατά συνέπεια, των περιπτώσεων χρήσης. Τέλος, το μοντέλο ελέγχου χρησιμοποιείται για τον έλεγχο του λογισμικού και επαληθεύει την ικανοποίηση των απαιτήσεων που περιγράφονται στο μοντέλο περιπτώσεων χρήσης.

## Σύνοψη ενότητας

---

*Κεντρική θέση στην ενοποιημένη προσέγγιση κατέχει η έννοια της περίπτωσης χρήσης. Μια περίπτωση χρήσης είναι μια αλληλουχία ενεργειών που εκτελεί το λογισμικό αλληλεπιδρώντας με το περιβάλλον του, προκειμένου να ικανοποιήσει μία λειτουργική απαίτηση. Το περιβάλλον του λογισμικού, δηλαδή οι χρήστες του, καθώς και τα εξωτερικά συστήματα με τα οποία αυτό αλληλεπιδρά, αποτελεί τους χειριστές αυτού. Για την περιγραφή των περιπτώσεων χρήσης χρησιμοποιούνται κατάλληλα δομημένα πρότυπα σε μορφή κειμένου, καθώς και διαγράμματα δραστηριοτήτων UML, τα οποία εντάσσονται σε μια τροποποιημένη μορφή του εγγράφου προδιαγραφών των απαιτήσεων από το λογισμικό. Όλα τα συστατικά τα οποία παράγονται κατά τον κύκλο ζωής του λογισμικού αναφέρονται σε κάποια περίπτωση χρήσης.*

## Άσκηση 2/Κεφάλαιο 8

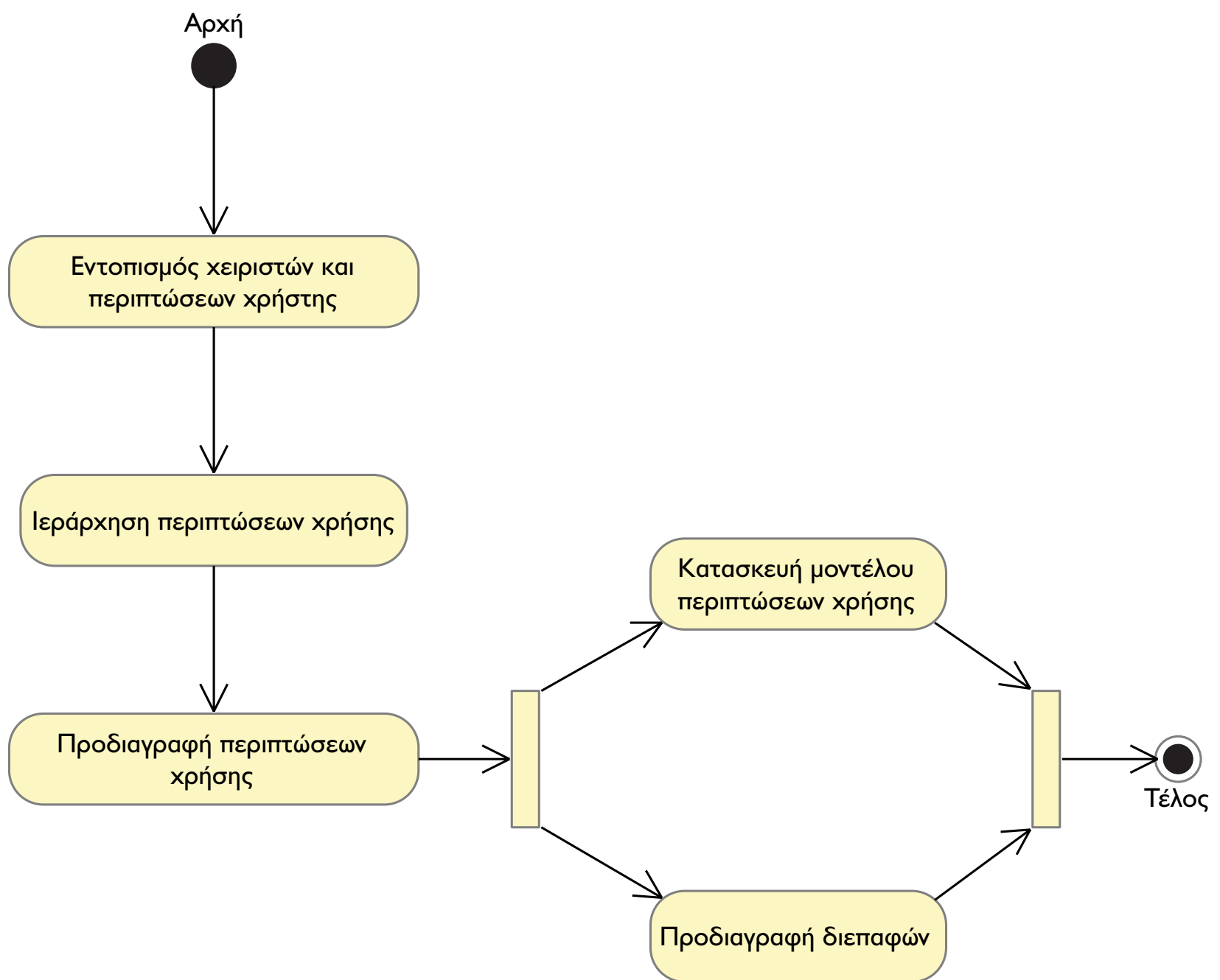
Παρατηρώντας το Σχήμα 8.9, είναι μάλλον αναπόφευκτο να αναγνωρίσουμε κάποια ομοιότητα με το μοντέλο κύκλου ζωής του καταρράκτη (Κεφάλαιο 2), γεγονός που, αν ισχύει, μάλλον δημιουργεί συγχύσεις σχετικά με τη διαφοροποίηση της αντικειμενοστρεφούς ανάπτυξης λογισμικού με την ενοποιημένη προσέγγιση. Μπορείτε να ξεκαθαρίσετε τις συγχύσεις αυτές; Για να καθοδηγηθείτε στην απάντηση, μπορείτε να συνδυάσετε το Σχήμα 8.9 με το Σχήμα 8.1 και γενικά με όσα αναφέρονται στην Ενότητα 8.1.

## ΕΝΟΤΗΤΑ 8.3. ΠΡΟΣΔΙΟΡΙΣΜΟΣ ΤΩΝ ΛΕΙΤΟΥΡΓΙΚΩΝ ΑΠΑΙΤΗΣΕΩΝ ΩΣ ΠΕΡΙΠΤΩΣΕΩΝ ΧΡΗΣΗΣ

Σε όλες τις φιλοσοφίες ανάπτυξης λογισμικού το πρόβλημα προσδιορισμού των απαιτήσεων είναι κυρίαρχο και η αντιμετώπισή του αποτελεί πρόκληση για τον κατασκευαστή, όντας η πιο κρίσιμη και δημιουργική εργασία στην ανάπτυξη. Εξαίρεση δεν αποτελεί η ενοποιημένη προσέγγιση, η οποία δεν διαθέτει κάποιο «μαγικό» τρόπο προσδιορισμού των απαιτήσεων από το λογισμικό. Η αντιμετώπιση του προβλήματος παραμένει ποιοτικά η ίδια με αυτή που αναφέρθηκε στο Κεφάλαιο 4.

Η αναφορά του θέματος στο σημείο αυτό δεν γίνεται με σκοπό την επανάληψη αλλά την εξειδίκευση της διαδικασίας προσδιορισμού των απαιτήσεων από το λογισμικό, ώστε να περιλαμβάνονται οι έννοιες που εισάγει η ενοποιημένη προσέγγιση, δηλαδή οι περιπτώσεις χρήσης, οι χειριστές και το μοντέλο περιπτώσεων χρήσης. Είσοδο στην εργασία αυτή αποτελεί ο ορισμός του προβλήματος καθώς και υλικό συνεντεύξεων με τον πελάτη. Ασάφειες ή ελλείψεις που υπάρχουν στη διατύπωση του προβλήματος ή στην αρχική καταγραφή των απαιτήσεων που έχουμε στη διάθεσή μας μεταφέρονται και στον προσδιορισμό και την προδιαγραφή των χειριστών και των περιπτώσεων χρήσης. Στο Σχήμα 8.10 φαίνεται η ροή των εργασιών κατά τον προσδιορισμό των απαιτήσεων ως περιπτώσεων χρήσης με διάγραμμα δραστηριότητας UML.

**Σχήμα 8.10** Ροή εργασιών κατά τον προσδιορισμό των απαιτήσεων ως περιπτώσεων χρήσης με διάγραμμα δραστηριότητας κατά την UML.



Το πρώτο βήμα είναι ο προσδιορισμός των χειριστών, δηλαδή ο καθορισμός του περιβάλλοντος λειτουργίας του λογισμικού. Τόσο ο καθορισμός των χειριστών που αντιστοιχούν σε εξωτερικά συστήματα όσο και αυτών που αντιστοιχούν σε χρήστες του λογισμικού γίνεται με βάση όσα είναι γνωστά στο σημείο αυτό και μπορούν να ωριμάσουν σε επόμενο κύκλο ανάπτυξης. Για παράδειγμα, είναι σύνηθες το φαινόμενο να εντοπίζονται στην αρχή περισσότεροι χειριστές που αντιστοιχούν σε χρήστες – φυσικά πρόσωπα απ' ό,τι πραγματικά χρειάζονται, οι οποίοι στη συνέχεια θα συγχωνευτούν.

Έχοντας μια εικόνα για τους χειριστές του συστήματος, μπορούμε να καθορίσουμε τις περιπτώσεις χρήσης. Καθεμία από αυτές θα πρέπει να συσχετίζεται με τουλάχιστον ένα χειριστή, πραγματοποιώντας μια εργασία χρήσιμη σε αυτόν. Ένας τρόπος προσδιορισμού των περιπτώσεων χρήσης είναι να παίρνουμε έναν έναν τους χειριστές και να εξαντλούμε τις απαιτήσεις τους σε εργασίες που πρέπει να πραγματοποιεί το λογισμικό. Αν μετά το τέλος της διαδικασίας αυτής διαπιστώσουμε ότι υπάρχουν εργασίες που γνωρίζουμε ότι πρέπει να κάνει το λογισμικό αλλά δεν έχουν απεικονιστεί ως περιπτώσεις χρήσης, τότε είτε θα πρέπει να ορίσουμε νέους χειριστές είτε να τις αντιστοιχίσουμε στους υπάρχοντες.

Η εργασία της ιεράρχησης των περιπτώσεων χρήσης είναι αντίστοιχη με την εργασία της ιεράρχησης των λειτουργικών απαιτήσεων. Το αποτέλεσμα της εργασίας αυτής πρέπει να απαντά στο ερώτημα: «Ποιες περιπτώσεις χρήσης είναι οι πιο κρίσιμες και επιθυμητές;». Με τον τρόπο αυτό υποστηρίζεται ο προγραμματισμός των εργασιών που θα γίνουν κατά τους κύκλους ανάπτυξης, ο οποίος λαμβάνει ασφαλώς υπόψη του και άλλα, μη τεχνικά ζητήματα, όπως διαθεσιμότητα προσωπικού, χρονοδιάγραμμα και προϋπολογισμός.

Ακολούθως καθορίζονται οι λεπτομέρειες για κάθε περίπτωση χρήσης με χρήση κειμένου που ακολουθεί τη δομή που δόθηκε στο Σχήμα 8.6 και δημιουργείται το μοντέλο περιπτώσεων χρήσης. Παράλληλα, μπορεί να καθορίζεται η διεπαφή με το χρήστη. Όταν δε περατωθούν οι εργασίες αυτές, μπορεί να θεωρείται περατωμένη η εργασία του προσδιορισμού των απαιτήσεων.

Ανάλογα με το μέγεθος του έργου καθορίζεται και η έκταση εφαρμογής της εργασίας του προσδιορισμού των απαιτήσεων που περιγράψαμε. Για μικρά έργα, όπως αυτό της δικής μας μελέτης περίπτωσης, η εργασία αυτή γίνεται σε έναν κύκλο ανάπτυξης. Μεγαλύτερα έργα απαιτούν την κατάτμηση σε μικρότερα τμήματα και την εκτέλεση καθενός από αυτά στον δικό του κύκλο ανάπτυξης. Στην περίπτωση αυτή, το μοντέλο περιπτώσεων χρήσης εμπλουτίζεται καθώς οι κύκλοι ανάπτυξης διαδέχονται ο ένας τον άλλο τόσο σε έκταση (πλήθος περιπτώσεων χρήσης) όσο και σε βάθος (λεπτομέρεια στην προδιαγραφή τους). Η διαχείριση και η διαρκής ενημέρωση του πλήθους των εγγράφων και των διαγραμμάτων που αναφέραμε είναι μια δύσκολη εργασία που καλό είναι να υποστηρίζεται από κάποιο εργαλείο.

## Σύνοψη ενότητας

*Οι λειτουργικές απαιτήσεις από το λογισμικό στην ενοποιημένη προσέγγιση προσδιορίζονται ως περιπτώσεις χρήσης. Το σύνολο των περιπτώσεων χρήσης αποτελεί το μοντέλο περιπτώσεων χρήσης. Η αντιμετώπιση του προβλήματος καθορισμού των περιπτώσεων χρήσης είναι παρόμοια με αυτό του καθορισμού των λειτουργικών απαιτήσεων.*

### Δραστηριότητα 4/Κεφάλαιο 8

Δώστε ένα διάγραμμα δραστηριότητας που περιγράφει την εργασία καθορισμού χειριστών και περιπτώσεων χρήσης, η οποία περιγράφηκε στην Ενότητα 3.3. Θα χρειαστεί να διαβάσετε με προσοχή τα αναφερόμενα στις δύο πρώτες παραγράφους που ακολουθούν το Σχήμα 8.10 και να συμβουλευτείτε τα σύμβολα της UML για τα διαγράμματα δραστηριότητας (Σχήμα 8.8).



## Μελέτη περίπτωσης

Θα επανέλθουμε στη μελέτη περίπτωσης που μας απασχόλησε στο Κεφάλαιο 4, αντιμετωπίζοντας αυτήν τη φορά την ανάπτυξή της με την αντικειμενοστρεφή τεχνολογία, σύμφωνα με την ενοποιημένη προσέγγιση. Στη συνέχεια παραθέτουμε τον ορισμό του προβλήματος και τις απαιτήσεις από το λογισμικό που εντοπίσαμε κατά την αναφορά μας στα πρώτα κεφάλαια.

### **«Επίκουρος»: Μια εφαρμογή λογισμικού για την υποστήριξη της γραμματείας εκπαιδευτικού φορέα.**

Ο πελάτης μας είναι υπεύθυνος για τη λειτουργία της γραμματείας ενός υποθετικού εκπαιδευτικού φορέα. Λόγω του πλήθους των σπουδαστών, των καθηγητών και των μαθημάτων, του όγκου και της πολυπλοκότητας των εργασιών υποστήριξης (αρχείου, εγγραφών, κ.ά.), είναι αναγκαία η χρήση μιας εφαρμογής λογισμικού, την οποία ο πελάτης αποφασίζει να ονομάσει «Επίκουρος» και μας αναθέτει την ανάπτυξή της.

Η εφαρμογή θα πρέπει να τηρεί αρχεία σπουδαστών, καθηγητών, μαθημάτων, εγγραφής σε μαθήματα, καθώς και αποτελέσματα βαθμολογίας. Η εφαρμογή θα πρέπει να εκτυπώνει καταστάσεις σπουδαστών, καθηγητών, μαθημάτων και βαθμολογίας με κριτήρια που θα δίνει ο χρήστης. Η εφαρμογή δεν θα πρέπει να επιτρέπει τη διαγραφή ενός σπουδαστή ή καθηγητή από το αρχείο, αν αυτός έχει εγγραφεί ή διδάξει μάθημα αντίστοιχα. Το περιβάλλον λειτουργίας θα είναι ένας αυτόνομος ηλεκτρονικός υπολογιστής με Windows.

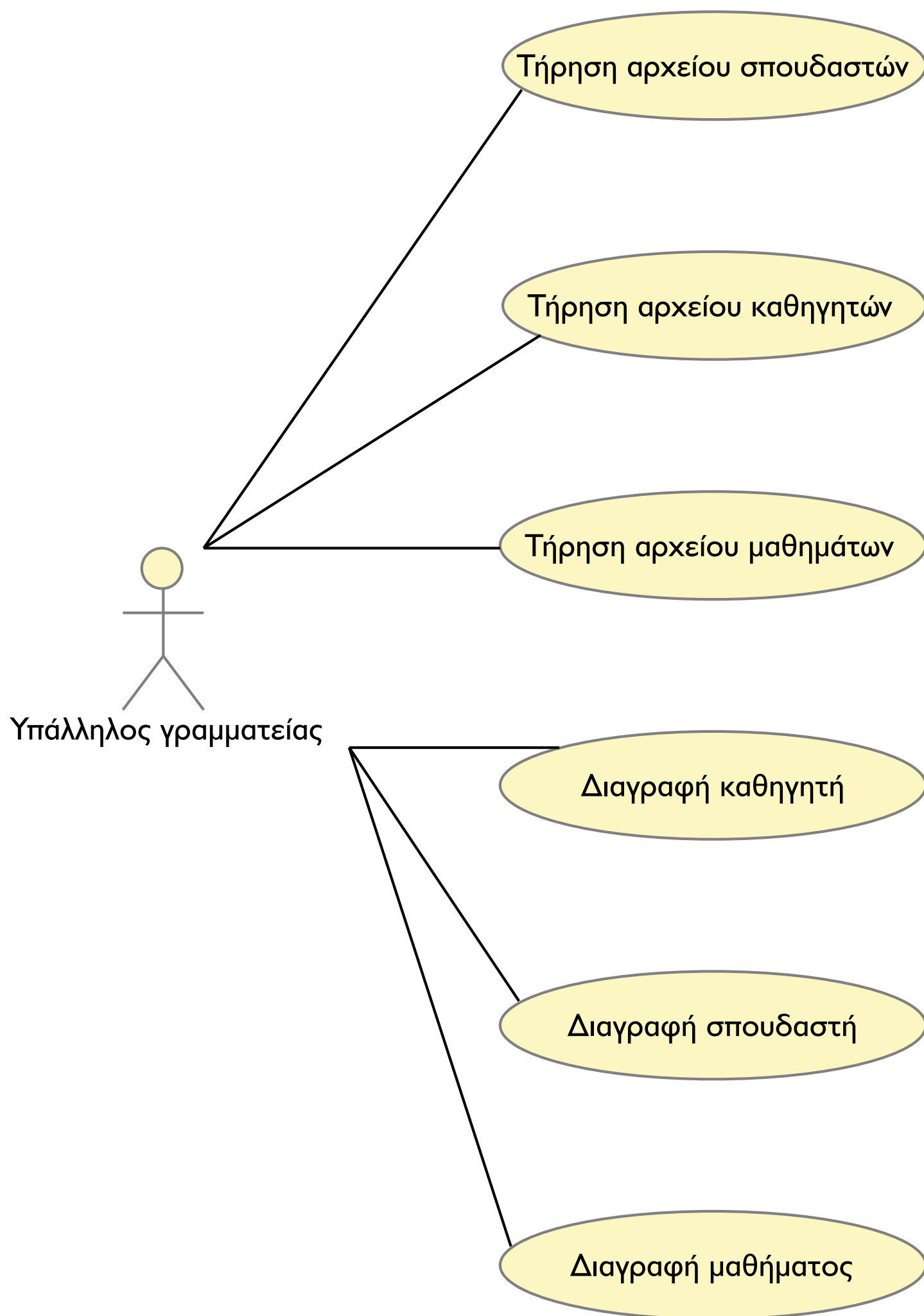
### Απαιτήσεις από το λογισμικό

1. Ο «Επίκουρος» θα τρέχει σε αυτόνομο υπολογιστή κάτω από το λειτουργικό σύστημα Windows. Δεν απαιτείται σύνδεση σε δίκτυο.
2. Ζητείται η τήρηση αρχείων μαθητών, καθηγητών και μαθημάτων.
3. Κάθε μάθημα διδάσκεται από έναν καθηγητή σε κάθε ακαδημαϊκό έτος.
4. Κάθε σπουδαστής μπορεί να εγγράφεται σε κάθε μάθημα όσες φορές θέλει.

5. Κάθε σπουδαστής αξιολογείται σε μαθήματα στα οποία έχει εγγραφεί. Η αξιολόγηση αυτή μπορεί να γίνεται περισσότερες από μία φορές τόσο κατά τη διάρκεια του ακαδημαϊκού έτους (ενδιάμεση εξέταση) όσο και με τελικό γραπτό.
6. Δεν πρέπει να επιτρέπεται η καταχώρηση βαθμολογίας σε μάθημα στο οποίο δεν έχει γίνει εγγραφή.
7. Επιτρέπεται η διαγραφή σπουδαστή μόνο αν δεν έχει εγγραφεί σε κανένα μάθημα.
8. Επιτρέπεται η διαγραφή καθηγητή μόνο αν δεν έχει διδάξει κανένα μάθημα.
9. Επιτρέπεται η διαγραφή μαθήματος μόνο αν δεν έχουν υπάρξει εγγραφές ή εξετάσεις που να το αφορούν.
10. Ζητείται αλφαβητική εκτύπωση ολόκληρου του αρχείου των σπουδαστών.
11. Ζητείται αλφαβητική εκτύπωση των εγγεγραμμένων σε κάθε μάθημα σπουδαστών.
12. Ζητείται αλφαβητική εκτύπωση ολόκληρου του αρχείου καθηγητών.
13. Ζητείται αλφαβητική εκτύπωση της βαθμολογίας σε κάθε μάθημα.
14. Ζητείται η εκτύπωση της βαθμολογίας όλων των μαθημάτων για κάποιο συγκεκριμένο σπουδαστή.

Θα κατασκευάσουμε ένα διάγραμμα περιπτώσεων χρήσης που αντιστοιχεί στις απαιτήσεις 2, 7, 8 και 9. Παρατηρήστε ότι για τη λειτουργική απαίτηση 2, η οποία αποτελεί σύζευξη τριών λειτουργικών απαιτήσεων, ορίζουμε τρεις περιπτώσεις χρήσης. Επίσης, προς το παρόν, θα θεωρήσουμε ότι έχουμε έναν χειριστή, ο οποίος αντιστοιχεί στους υπαλλήλους της γραμματείας. Το διάγραμμα αυτό φαίνεται στο Σχήμα 8.11.

**Σχήμα 8.11** Ένα διάγραμμα περιπτώσεων χρήσης για μερικές από τις λειτουργικές απαιτήσεις του λογισμικού «Επίκουρος».



Στη συνέχεια θα δώσουμε την αναλυτική περιγραφή των περιπτώσεων χρήσης «τήρηση αρχείου σπουδαστών» και «διαγραφή σπουδαστή», σύμφωνα με το πρότυπο που δώσαμε στο Σχήμα 8.6.

## ΠΡΟΔΙΑΓΡΑΦΗ ΠΕΡΙΠΤΩΣΗΣ ΧΡΗΣΗΣ

### 1. Τίτλος περίπτωσης χρήσης

*Τήρηση αρχείου σπουδαστών.*

### 2. Σύντομη περιγραφή

*Η εφαρμογή εμφανίζει μια φόρμα διαλόγου μέσω της οποίας ο χρήστης ενημερώνει το αρχείο σπουδαστών.*

### 3. Ροή γεγονότων

#### 3.1 Βασική ροή

- 1. Ο χειριστής «χειριστής γραμματείας» επιλέγει από το μενού της εφαρμογής την «τήρηση αρχείου σπουδαστών».*
- 2. Ο «Επίκουρος» ανοίγει το αρχείο σπουδαστών.*
- 3. Ο «Επίκουρος» εμφανίζει φόρμα διαλόγου με τα πεδία που περιλαμβάνονται στο αρχείο, καθώς και δύο κουμπιά (buttons) με χαρακτηρισμούς «αποδοχή» και «ακύρωση».*
- 4. Ο χειριστής δίνει τα στοιχεία σπουδαστή που περιέχονται στη φόρμα και πατάει το κουμπί «αποδοχή».*
- 5. Ο «Επίκουρος» ελέγχει την εγκυρότητα των στοιχείων.*
- 6. Ο «Επίκουρος» εισάγει μια νέα εγγραφή στο αρχείο σπουδαστών.*
- 7. Ο έλεγχος επανέρχεται στο βήμα 2.*

#### 3.2 Εναλλακτικές ροές

##### 3.2.1 Εναλλακτική ροή 1

*4α. Ο χειριστής επιλέγει «ακύρωση».*

*5α. Ο «Επίκουρος» κλείνει τη φόρμα και τερματίζει την εργασία.*

##### 3.2.2 Εναλλακτική ροή 2

6α. Τα στοιχεία που δόθηκαν είναι ελλιπή.

7α. Ο «Επίκουρος» εμφανίζει μήνυμα στον χρήστη και επανέρχεται στο βήμα 2.

#### **4. Μη λειτουργικές απαιτήσεις**

*Δεν υπάρχουν για αυτή την περίπτωση χρήσης.*

#### **5. Κατάσταση εισόδου**

*Δεν υπάρχουν ιδιαίτερες απαιτήσεις εισόδου στην περίπτωση χρήσης.*

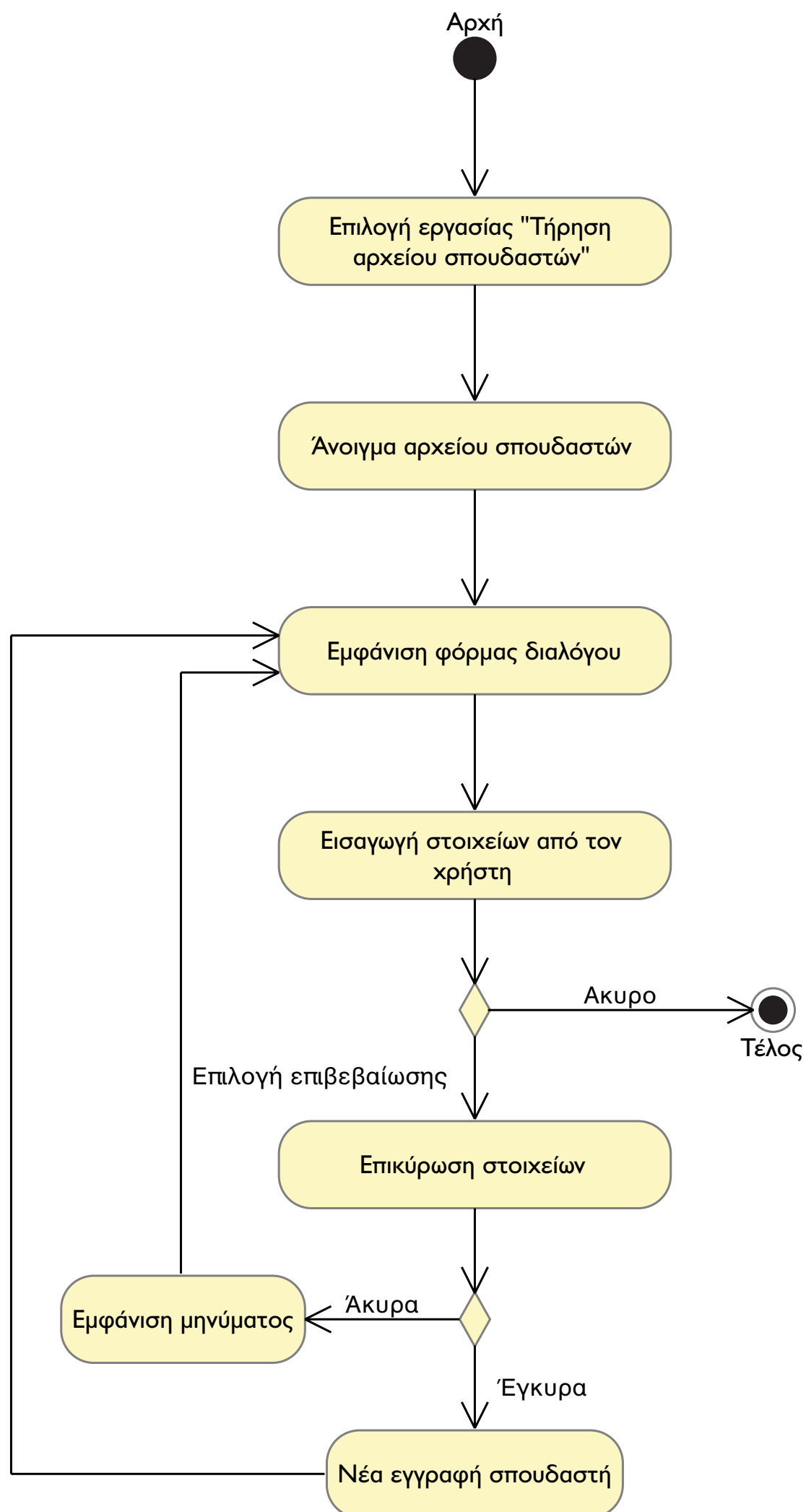
#### **6. Κατάσταση εξόδου**

*Εχουν προστεθεί 0-N νέες εγγραφές στο αρχείο σπουδαστών.*

### **Διάγραμμα δραστηριότητας περίπτωσης χρήσης**

Στο Σχήμα 8.12 που ακολουθεί δίνεται το διάγραμμα δραστηριότητας για την παραπάνω περιγραφή της ροής εργασιών της περίπτωσης χρήσης. Παρατηρήστε ότι λόγω σχετικής απλότητας τόσο η κύρια όσο και οι εναλλακτικές ροές παριστάνονται με τη βοήθεια ενός και μόνο διαγράμματος.

**Σχήμα 8.12** Το διάγραμμα δραστηριότητας για την περίπτωση χρήσης «διαχείριση αρχείου σπουδαστών» του λογισμικού «Επίκουρος».



# ΠΡΟΔΙΑΓΡΑΦΗ ΠΕΡΙΠΤΩΣΗΣ ΧΡΗΣΗΣ

## I. Τίτλος περίπτωσης χρήσης

Διαγραφή σπουδαστή.

## 2. Σύντομη περιγραφή

Η εφαρμογή εμφανίζει μια φόρμα η οποία περιέχει αλφαβητικά ταξινομημένη λίστα όλων των σπουδαστών. Ο χειριστής επιλέγει τον σπουδαστή που θέλει να διαγράψει και επιβεβαιώνει τη διαγραφή. Η εργασία επαναλαμβάνεται μέχρι ο χειριστής να πατήσει «ακύρωση».

## 3. Ροή γεγονότων

### 3.1 Βασική ροή

1. Ο χειριστής «χειριστής γραμματείας» επιλέγει από το μενού της εφαρμογής την εργασία «διαγραφή σπουδαστών».
2. Ο «Επίκουρος» εντοπίζει τους σπουδαστές των οποίων η διαγραφή επιτρέπεται, δηλαδή αυτούς που δεν έχουν εγγραφεί σε κανένα μάθημα, και τους τοποθετεί σε αλφαβητικά ταξινομημένη λίστα.
3. Η εφαρμογή εμφανίζει φόρμα που περιέχει τη λίστα που δημιουργήθηκε στο βήμα 2, καθώς και δύο κουμπιά (buttons) με χαρακτηρισμούς «διαγραφή» και «ακύρωση».
4. Αν ο χειριστής επιλέξει «ακύρωση», τότε η εργασία τερματίζεται.
5. Ο χειριστής επιλέγει έναν σπουδαστή από τη λίστα και πατάει το κουμπί «διαγραφή».
6. Η εφαρμογή εμφανίζει παράθυρο διαλόγου με την ερώτηση «Επιβεβαίωση;» και δύο κουμπιά, «Ναι» και «Όχι».
7. Αν ο χειριστής επιλέξει «Ναι», η επιλεγμένη εγγραφή διαγράφεται από το αρχείο σπουδαστών, διαφορετικά δεν συμβαίνει τίποτε.
8. Ο έλεγχος επανέρχεται στο βήμα 2.

## 3.2 Εναλλακτικές ροές

### 3.2.1 Εναλλακτική ροή I

3α. Δεν υπάρχει κανένας σπουδαστής του οποίου η διαγραφή να επιτρέπεται.

3β. Ο «Επίκουρος» εμφανίζει παράθυρο διαλόγου που ενημερώνει σχετικά τον χειριστή και, αφού αυτός πατήσει «αποδοχή», η εργασία τερματίζεται.

## 4. Μη λειτουργικές απαιτήσεις

Δεν υπάρχουν γι' αυτή την περίπτωση χρήσης.

## 5. Κατάσταση εισόδου

Δεν υπάρχουν ιδιαίτερες απαιτήσεις εισόδου στην περίπτωση χρήσης.

## 6. Κατάσταση εξόδου

Έχουν διαγραφεί 0-N εγγραφές από το αρχείο σπουδαστών.

Παρατήρηση: Είναι πιθανό από την εμπειρία σας ως χρήστες εφαρμογών λογισμικού να έχετε διαφορετική άποψη για τον τρόπο συμπεριφοράς της εφαρμογής κατά την εκτέλεση των περιπτώσεων χρήσης που περιγράφονται στο σημείο αυτό. Αυτό είναι όχι μόνο φυσιολογικό αλλά και επιθυμητό. Θα σας θυμίσουμε, όμως, ότι το ζητούμενο από την ενασχόλησή μας με την εφαρμογή «Επίκουρος» στο βιβλίο αυτό δεν είναι να δώσουμε την καλύτερη λύση ή να την αναπτύξουμε πλήρως και σε βαθμό που να μπορεί να χρησιμοποιηθεί σε πραγματικό περιβάλλον, αλλά να επιδείξουμε την εφαρμογή της αντικειμενοστρεφούς φιλοσοφίας σύμφωνα με την ενοποιημένη προσέγγιση. Μπορείτε (και ενθαρρύνεστε) να εμβαθύνετε σε όποιο βαθμό θέλετε κατά την ενασχόλησή σας με τις δραστηριότητες.



## Δραστηριότητα 5/Κεφάλαιο 8

Θα συμπληρώσουμε την αναφορά μας στη μελέτη περίπτωσης προσθέτοντας δύο ακόμη απαιτήσεις στη λίστα της Ενότητας 3.2 ως ακολούθως:

15. Ο σπουδαστής μπορεί να χρησιμοποιεί ένα τερματικό του συστήματος για να βλέπει ο ίδιος τη βαθμολογία του σε όλα τα μαθήματα.

16. Ο διδάσκων ενός μαθήματος είναι δυνατόν να καταχωρεί ο ίδιος τη βαθμολογία των σπουδαστών σε αυτό.

Εντοπίστε όλες τις περιπτώσεις χρήσης και τους χειριστές του λογισμικού «Επίκουρος» και δώστε ολόκληρο το διάγραμμα περιπτώσεων χρήσης της εφαρμογής.

## Άσκηση 3/Κεφάλαιο 8

Μπορείτε να παρατηρήσετε ότι οι περιπτώσεις χρήσης που ορίσαμε δεν αντιστοιχούν μία προς μία στις απαιτήσεις I-I6 του λογισμικού «Επίκουρος», όπως οι τελευταίες που αναφέρθηκαν στη μελέτη περίπτωσης. Ήδη, για παράδειγμα, εντοπίσαμε ότι η απαίτηση 2 αναλύθηκε σε τρεις περιπτώσεις χρήσης.

Προσπαθήστε να εντοπίσετε ποιες από τις απαιτήσεις που δόθηκαν στον ορισμό της μελέτης περίπτωσης δεν αντιστοιχούν σε περιπτώσεις χρήσης και να εντοπίσετε το γιατί.

## Δραστηριότητα 6/Κεφάλαιο 8

Κατασκευάστε το διάγραμμα δραστηριότητας για την περίπτωση χρήσης «διαγραφή σπουδαστή», όπως αυτή περιγράφηκε στη μελέτη περίπτωσης που προηγήθηκε.

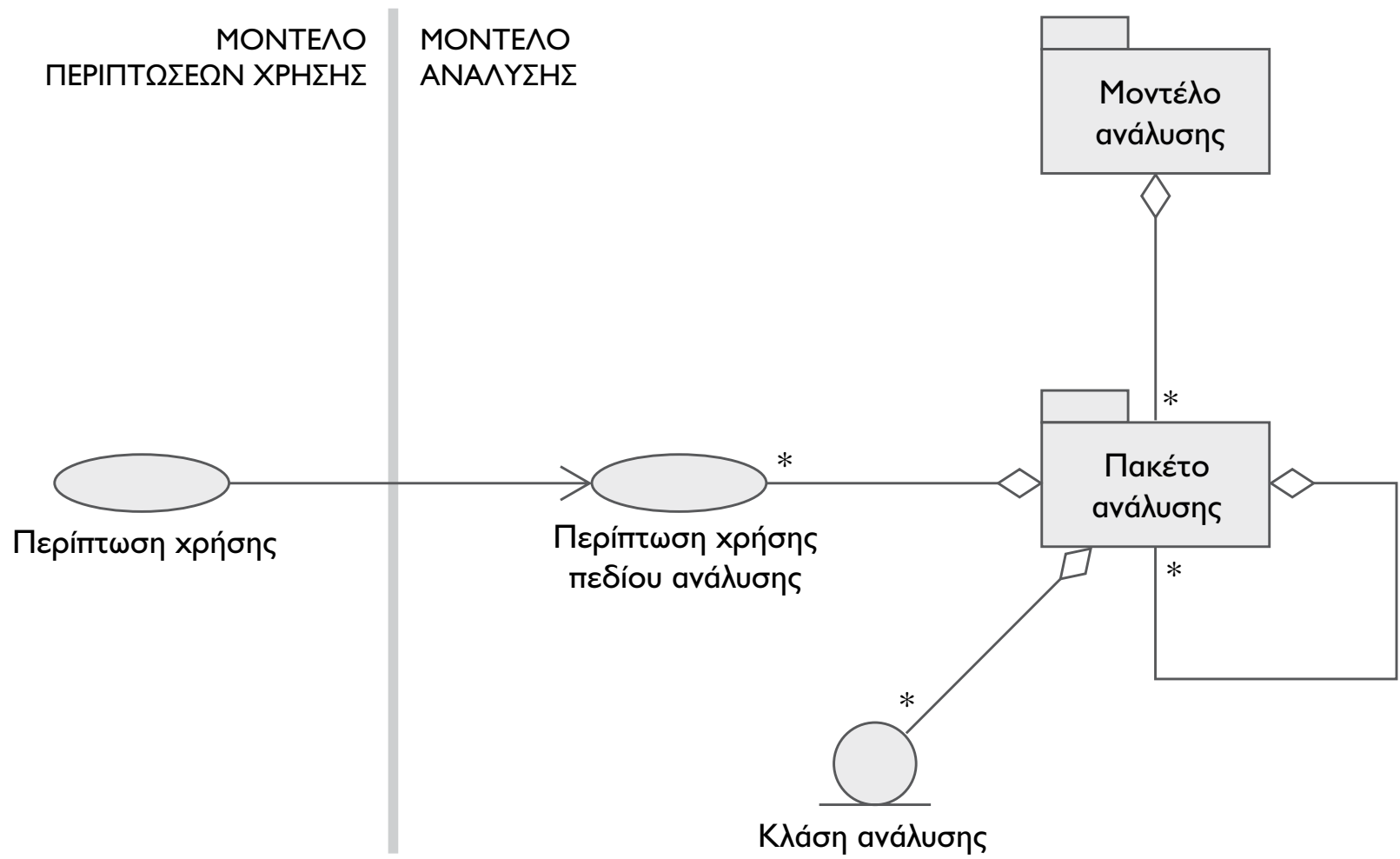
## ΕΝΟΤΗΤΑ 8.4. ΑΠΟ ΤΙΣ ΠΕΡΙΠΤΩΣΕΙΣ ΧΡΗΣΗΣ ΣΤΟ ΜΟΝΤΕΛΟ ΑΝΑΛΥΣΗΣ

Στην ενοποιημένη προσέγγιση οι περιπτώσεις χρήσης είναι το μέσο για την αποτύπωση των απαιτήσεων του πελάτη κατά τρόπο ώστε αυτές να αποτελούν σημείο αναφοράς για τη συνέχεια της ανάπτυξης. Όπως αναφέραμε, οφείλουν να είναι κατανοητές και από τον πελάτη, γεγονός που αναγκαστικά τους αφαιρεί τη λεπτομέρεια που απαιτείται από την πλευρά ενός κατασκευαστή, ο οποίος δεν σκέφτεται μόνο σε όρους λειτουργικότητας όπως ο πελάτης, αλλά και σε όρους υλοποίησης. Κατά την ανάλυση, με εναρκτήριο σημείο το μοντέλο περιπτώσεων χρήσης, κατασκευάζεται το μοντέλο ανάλυσης.

### 8.4.1. Το μοντέλο ανάλυσης

Ένα μοντέλο ανάλυσης είναι ένα μοντέλο παράστασης λογισμικού το οποίο αποτελείται από τουλάχιστον ένα πακέτο ανάλυσης (analysis package), το οποίο μπορεί να περιλαμβάνει και άλλα πακέτα τα οποία με τη σειρά τους περιλαμβάνουν κλάσεις ανάλυσης ή και άλλα πακέτα κ.ο.κ. Κάθε περίπτωση χρήσης του μοντέλου περιπτώσεων χρήσης αντιστοιχίζεται σε μια περίπτωση χρήσης πεδίου ανάλυσης του μοντέλου ανάλυσης, η οποία με τη σειρά της αντιστοιχίζεται σε κάποιο πακέτο ανάλυσης (Σχήμα 8.13).

**Σχήμα 8.13** Από τις περιπτώσεις χρήσης στο μοντέλο ανάλυσης.



Το μοντέλο ανάλυσης εκλεπτύνει τις απαιτήσεις του μοντέλου περιπτώσεων χρήσης στη γλώσσα των κατασκευαστών, έτσι ώστε να μπορούν να εντοπιστούν αρχιτεκτονικά στοιχεία του λογισμικού που απαιτούνται για τη συνέχεια της ανάπτυξης, δηλαδή να καθοριστούν οι κλάσεις που θα αποτελέσουν την εφαρμογή λογισμικού, η ομαδοποίησή τους και οι συσχετίσεις μεταξύ αυτών. Αυτά, ωστόσο, τα χαρακτηριστικά είναι μια πρώτη εκδοχή της κατασκευαστικής δομής του λογισμικού, η οποία αναμένεται να μεταβληθεί κατά τη σχεδίαση. Εκεί, προκειμένου να ικανοποιηθούν και οι μη λειτουργικές απαιτήσεις, θα προστεθούν κατασκευαστικές λεπτομέρειες και χαρακτηριστικά του περιβάλλοντος ανάπτυξης και λειτουργίας, τα οποία με κανένα τρόπο δεν απασχολούν την εργασία της ανάλυσης.

Κάθε πακέτο ανάλυσης περιλαμβάνει κλάσεις οι οποίες υλοποιούν τη λειτουργική συμπεριφορά μίας ή περισσότερων περιπτώσεων χρήσης. Με τον τρόπο αυτό μεταβαίνουμε από την περιγραφή της απαίτησης σε μια πρώτη περιγραφή υλοποίησης με όρους δομικών μονάδων λογισμικού, δηλαδή των κλάσεων ανάλυσης. Επίσης, κάθε πακέτο ανάλυσης μπορεί να περιλαμβάνει και άλλα πακέτα ανάλυσης και με τον τρόπο αυτό μπορεί να γίνει πιο αναγνώσιμο και εύχρηστο το μοντέλο ανάλυσης.

#### Άσκηση 4/Κεφάλαιο 8

**Τι εννοούμε με τη φράση «να εντοπιστούν αρχιτεκτονικά στοιχεία του λογισμικού που απαιτούνται για τη συνέχεια της ανάπτυξης»; Ποια είναι τα τρία αρχιτεκτονικά στοιχεία που πρέπει να εντοπιστούν κατά την ανάλυση; Για ποιο λόγο η ανάλυση δεν ταυτίζεται με τη σχεδίαση στην αντικειμενοστρεφή ανάπτυξη λογισμικού με την ενοποιημένη προσέγγιση;**

### 8.4.2. Κλάσεις στο μοντέλο ανάλυσης

Οι κλάσεις ανάλυσης ορίζονται ώστε να πραγματοποιήσουν τα καθοριζόμενα από τις περιπτώσεις χρήσης. Αυτό σημαίνει ότι πεδία, μέθοδοι και σχέσεις δεν ορίζονται με την αυστηρότητα που απαιτείται για να είναι υλοποιήσιμες σε μια αντικειμενοστρεφή γλώσσα προγραμματισμού αλλά με σκοπό την απόδοση των ποιοτικών χαρακτηριστικών της κλάσης. Για τον λόγο αυτό είναι χρήσιμη μια διάκριση των κλάσεων ανάλυσης σε κατηγορίες, ώστε να είναι ευκολότερα αντιληπτά τα μοντέλα παράστασης λογισμικού τα οποία τις περιλαμβάνουν. Θα διακρίνουμε, λοιπόν, τις κλάσεις ανάλυσης σε συνοριακές (boundary), οντοτήτων (entity) και ελέγχου (control).

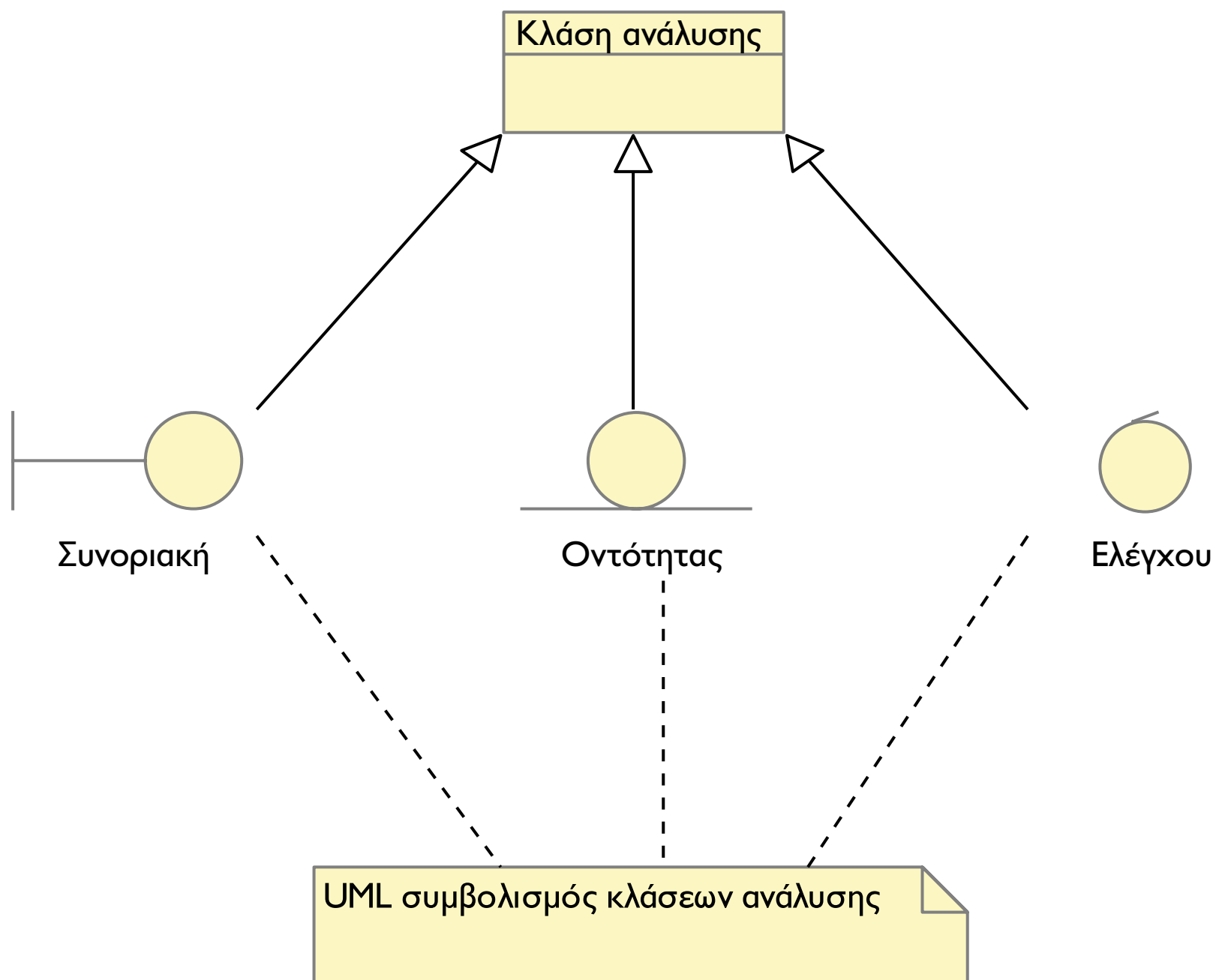
Μια συνοριακή κλάση ορίζεται για την παράσταση της αλληλεπίδρασης του λογισμικού με τους χειριστές του και σχετίζεται τουλάχιστον με έναν από αυτούς. Οι συνοριακές κλάσεις καθορίζουν πλήρως την επικοινωνία του λογισμικού με το περιβάλλον του, δηλαδή τους χρήστες – φυσικά πρόσωπα και τα εξωτερικά συστήματα. Συνήθως αντιστοιχούν σε παράθυρα διαλόγου, οδηγούς συσκευών, πρωτόκολλα επικοινωνιών κ.ά.

Μια κλάση οντοτήτων παριστάνει πληροφορίες, οντότητες και συμβάντα του πραγματικού κόσμου, οι οποίες είναι εντός του πεδίου ενδιαφέροντος της εφαρμογής λογισμικού. Με τέτοιες κλάσεις μοντελοποιούνται όλες οι έννοιες του πεδίου του προβλήματος, οι οποίες πρέπει να αποθηκεύονται μόνιμα, συνήθως σε κάποια βάση δεδομένων. Οι κλάσεις οντοτήτων μπορούμε να θεωρούμε ότι αντιστοιχούν στις οντότητες του μοντέλου οντοτήτων – συσχετίσεων με τη διαφορά ότι, εκτός από πεδία, μπορούν να περιλαμβάνουν και στοιχεία συμπεριφοράς, δηλαδή μεθόδους.

Μια κλάση ελέγχου αντιστοιχεί στον συντονισμό και τη διαχείριση δοσοληψιών (transactions) και γεγονότων (events), καθώς και στον έλεγχο ροής προγράμματος και την πραγματοποίηση υπολογισμών οι οποίοι δεν μπορούν να αποδοθούν ούτε σε συνοριακές ούτε σε κλάσεις οντοτήτων.

Στο Σχήμα 8.14 φαίνονται τα σύμβολα της UML για τις τρεις κατηγορίες κλάσεων στο μοντέλο ανάλυσης.

**Σχήμα 8.14** Συμβολισμοί UML για τις κλάσεις ανάλυσης.



## Άσκηση 5/Κεφάλαιο 8

**Κατατάξτε τις παρακάτω κλάσεις ανάλυσης στις τρεις κατηγορίες που αναφέρθηκαν.**

1. Διδάσκων
2. Σπουδαστής
3. Φόρμα εισαγωγής στοιχείων
4. Παράθυρο επιβεβαίωσης διαγραφής
5. Μάθημα
6. Διαχειριστής συστήματος backup
7. Βαθμολογία μαθήματος
8. Οδηγός αισθητήρα θερμοκρασίας
9. Διαχειριστής εκτυπώσεων συστήματος
10. Διαχειριστής τραπεζικών δοσοληψιών

### 8.4.3. Πακέτα ανάλυσης

Τα πακέτα ανάλυσης είναι ένα χρήσιμο εργαλείο για την ομαδοποίηση των συστατικών του μοντέλου ανάλυσης. Όπως φαίνεται στο Σχήμα 8.13, ένα πακέτο ανάλυσης περιέχει κλάσεις ανάλυσης και περιπτώσεις χρήσης πεδίου ανάλυσης. Με τη βοήθεια των πακέτων ανάλυσης επιμερίζεται το πρόβλημα της ανάλυσης σε μικρότερα και εφαρμόζεται το «διαίρει και βασίλευε». Αν όλες οι κλάσεις που προέκυπταν από την ανάλυση των περιπτώσεων χρήσης εμφανίζονταν μαζί, τότε το πρόβλημα της παράστασης και της διαχείρισης του μοντέλου ανάλυσης που θα προέκυπτε θα ήταν πολύ μεγάλο.

Με τη βοήθεια των πακέτων ανάλυσης, οι κλάσεις που προκύπτουν από την ανάλυση των περιπτώσεων χρήσης ομαδοποιούνται. Δεν υπάρχουν αδιαμφισβήτητα και μοναδικά κριτήρια ομαδοποίησης των κλάσεων ανάλυσης σε πακέτα. Η γενική αρχή είναι ότι σε ένα πακέτο ανήκουν σημασιολογικά



συναφείς κλάσεις όπως, για παράδειγμα, κλάσεις που υλοποιούν περιπτώσεις χρήσης που ανήκουν σε κάποια κατηγορία την οποία αυθαίρετα ορίζουμε ή κλάσεις που σχετίζονται με κάποιον συγκεκριμένο χειριστή. Στη μελέτη περίπτωσης «Επίκουρος» λόγου χάρη, όλες οι κλάσεις που σχετίζονται με τις περιπτώσεις χρήσης που αφορούν μαθητές μπορεί να ανήκουν σε ένα πακέτο.

Ένα άλλο κριτήριο διάκρισης των κλάσεων ανάλυσης σε πακέτα αποτελεί η δυνατότητα επαναχρησιμοποίησης αυτών. Αν κατά την ανάλυση ενός προβλήματος διαπιστώσουμε ότι υπάρχουν σε αυτό χαρακτηριστικά που είναι γενικά και απαντώνται και σε άλλα προβλήματα, είναι θεμιτό να θέλουμε να επαναχρησιμοποιήσουμε τα αποτελέσματα αυτής και για τα άλλα προβλήματα. Ένα υποσύνολο του μοντέλου ανάλυσης για το οποίο ισχύει το ενδεχόμενο επαναχρησιμοποίησης μπορεί να ορίσει ένα πακέτο ανάλυσης.

Τέλος, μπορούμε να ορίσουμε πακέτα με κριτήριο την περιγραφή διαφορετικών απόψεων (views) ενός συστήματος. Σύμφωνα με αυτή τη διάκριση μπορούν να υπάρχουν πακέτα που περιέχουν μόνο περιπτώσεις χρήσης που αφορούν χειριστές – φυσικά πρόσωπα και περιγράφουν την όψη της συμπεριφοράς του λογισμικού προς τους χρήστες του, πακέτα που περιέχουν συστατικά του πεδίου της υλοποίησης και αφορούν την όψη του συστήματος προς τους προγραμματιστές, πακέτα που περιέχουν συστατικά τα οποία περιγράφουν την ανάθεση εργασιών σε υπολογιστικές μονάδες και αφορούν την όψη της εγκατάστασης του συστήματος κ.ά.

## Σύνοψη ενότητας

---

*Το μοντέλο ανάλυσης είναι το πρώτο βήμα κατά τη μετάβαση από τις περιπτώσεις χρήσης στα δομικά συστατικά του λογισμικού τα οποία θα τις υλοποιήσουν. Περιέχει συνδέσμους με το μοντέλο περιπτώσεων χρήσης, καθώς και κλάσεις ανάλυσης οι οποίες αποτελούν την πρώτη εκδοχή συστατικών στοιχείων υλοποίησης λογισμικού, τα οποία εμφανίζονται στην ανάπτυξη, σύμφωνα με την ενοποιημένη προσέγγιση. Οι κλάσεις ανάλυσης διακρίνονται σε τρεις κατηγορίες, στις συνοριακές, στις κλάσεις οντοτήτων και στις κλάσεις ελέγχου.*

Οι συνοριακές κλάσεις αφορούν την αλληλεπίδραση του λογισμικού με το περιβάλλον του, οι κλάσεις οντοτήτων αντιστοιχούν σε έννοιες του πεδίου του προβλήματος, ενώ οι κλάσεις ελέγχου αφορούν γενικά τον έλεγχο ροής προγράμματος. Προκειμένου να είναι πιο εύκολα κατανοητό και διαχειρίσιμο, το μοντέλο ανάλυσης δομείται σε πακέτα ανάλυσης.

### Άσκηση 6/Κεφάλαιο 8

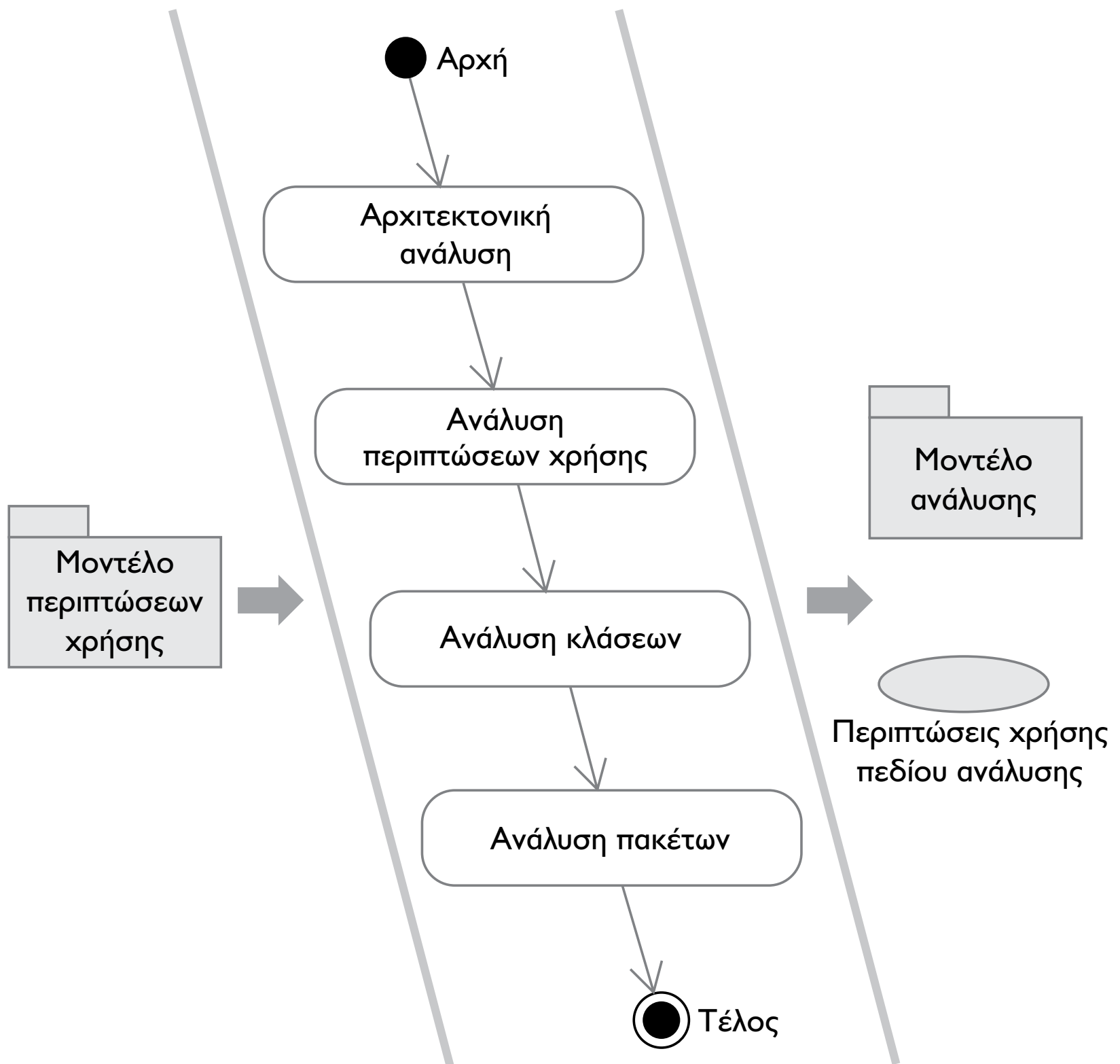
**Πώς ερμηνεύετε τη συσχέτιση του πακέτου ανάλυσης με τον εαυτό του μέσω της σχέσης συναρμολόγησης η οποία φαίνεται στο Σχήμα 8.13;**

## ΕΝΟΤΗΤΑ 8.5. ΒΗΜΑΤΑ ΣΤΗΝ ΑΝΑΛΥΣΗ

Αφού έχουμε αναφερθεί στα συστατικά στοιχεία του μοντέλου ανάλυσης, μπορούμε να προσεγγίσουμε τη γενική ακολουθία βημάτων που ακολουθούνται κατά την ανάλυση. Δεδομένα εισόδου στην εργασία αυτή είναι το μοντέλο περιπτώσεων χρήσης, το οποίο περιλαμβάνει την τεκμηρίωση που συνοδεύει τις περιπτώσεις χρήσης, όπως καθορίστηκε κατά την καταγραφή των λειτουργικών απαιτήσεων ως περιπτώσεις χρήσης. Αποτελέσματα της εργασίας είναι το μοντέλο ανάλυσης το οποίο περιέχει τα συστατικά στοιχεία που περιγράφηκαν στην Ενότητα 3.4, καθώς και το σύνολο των περιπτώσεων χρήσης πεδίου ανάλυσης το οποίο μπορεί να ιδωθεί ως γέφυρα μεταξύ του μοντέλου περιπτώσεων χρήσης και του μοντέλου σχεδίασης. Με τον τρόπο αυτό εξασφαλίζεται η ιχνηλασιμότητα των περιπτώσεων χρήσης που είναι αντιληπτές από τον πελάτη στη συνέχεια της ανάπτυξης, όπου εισάγονται έννοιες δομικών συστατικών λογισμικού χρήσιμες στον κατασκευαστή.

Στην αντικειμενοστρεφή ανάπτυξη με την ενοποιημένη προσέγγιση αναγνωρίζονται τέσσερα επιμέρους βήματα στην ανάλυση, τα οποία φαίνονται στο Σχήμα 8.15.

**Σχήμα 8.15** Βήματα στην ανάλυση με την ενοποιημένη προσέγγιση.



Κατά την αρχιτεκτονική ανάλυση διαγράφεται η δομή του μοντέλου ανάλυσης με τον καθορισμό των πακέτων ανάλυσης. Κατά την ανάλυση των περιπτώσεων χρήσης καθορίζονται οι κλάσεις ανάλυσης που είναι απαραίτητες έτσι ώστε να επιτευχθεί η απαιτούμενη λειτουργική συμπεριφορά κάθε περίπτωσης χρήσης. Κατά την ανάλυση κλάσεων καθορίζονται τα πεδία και οι μέθοδοι κάθε κλάσης, καθώς και οι συσχετίσεις της με άλλες κλάσεις ανάλυσης. Τέλος, κατά την ανάλυση πακέτων επιβεβαιώνεται η δόμηση του μοντέλου ανάλυσης και η ικανοποίηση των περιπτώσεων χρήσης που έχουν αντιστοιχηθεί στα πακέτα ανάλυσης. Στις υποενότητες που ακολουθούν περιγράφεται ο τρόπος εκτέλεσης των εργασιών αυτών σύμφωνα με την ενοποιημένη προσέγγιση.

### 8.5.1. Αρχιτεκτονική ανάλυση

Σκοπός της αρχιτεκτονικής ανάλυσης είναι ο καθορισμός των πακέτων του μοντέλου ανάλυσης. Με τη δόμηση του μοντέλου ανάλυσης σε πακέτα επιτυγχάνεται μια κατάτμηση του προβλήματος, η οποία είναι επιθυμητή για την αντιμετώπισή του σε μικρά τμήματα. Δεν είναι απαραίτητο και, συνήθως, δεν είναι δυνατό να καθοριστούν με την πρώτη όλα τα πακέτα ανάλυσης. Συνήθως, εντοπίζεται μια πρώτη δομή η οποία διαμορφώνεται με τον ορισμό νέων πακέτων σε πολλά επίπεδα, σε κάθε κύκλο ανάπτυξης.

Σε πρώτο επίπεδο, ο καθορισμός των πακέτων μπορεί να πραγματοποιηθεί με τα ακόλουθα κριτήρια:

- Ένα πακέτο αντιστοιχεί σε περιπτώσεις χρήσης που είναι σημασιολογικά συναφείς στο πεδίο του προβλήματος.
- Ένα πακέτο αντιστοιχεί σε περιπτώσεις χρήσης που σχετίζονται με έναν χειριστή.
- Ένα πακέτο είναι όσο το δυνατόν πιο ανεξάρτητο από τα υπόλοιπα.
- Ένα πακέτο περιγράφει μια όψη (view) του συστήματος.

Η εφαρμογή των κριτηρίων αυτών στην πράξη μπορεί, ασφαλώς, να γίνει με πολλούς τρόπους. Η σημασιολογική συνάφεια μπορεί να γίνεται με πολλούς τρόπους αντιληπτή και να προκύπτουν περισσότερα του ενός σύνολα

πακέτων ανάλυσης για το ίδιο πρόβλημα. Ο ορισμός πακέτων ανάλυσης με κριτήριο τη σημασιολογική συνάφεια μπορεί να είναι διαφορετικός από αυτόν που θα προκύψει με κριτήριο την αντιστοίχιση στον ίδιο χειριστή. Συνήθως, όταν ο χειριστής περιγράφει εξωτερικό σύστημα και όχι χρήστη – φυσικό πρόσωπο, η αντιστοίχιση πακέτου ανάλυσης στις περιπτώσεις χρήσης με τις οποίες αυτός σχετίζεται ικανοποιεί και το τρίτο κριτήριο, δηλαδή την ανεξαρτησία του πακέτου ανάλυσης που προκύπτει από τα υπόλοιπα. Γενικά, η ανεξαρτησία που αναφέρεται ως κριτήριο έχει το νόημα της ελαχιστοποίησης των επιπτώσεων μιας μεταβολής σε κάποιο από τα συστατικά ενός πακέτου ανάλυσης σε άλλα πακέτα.

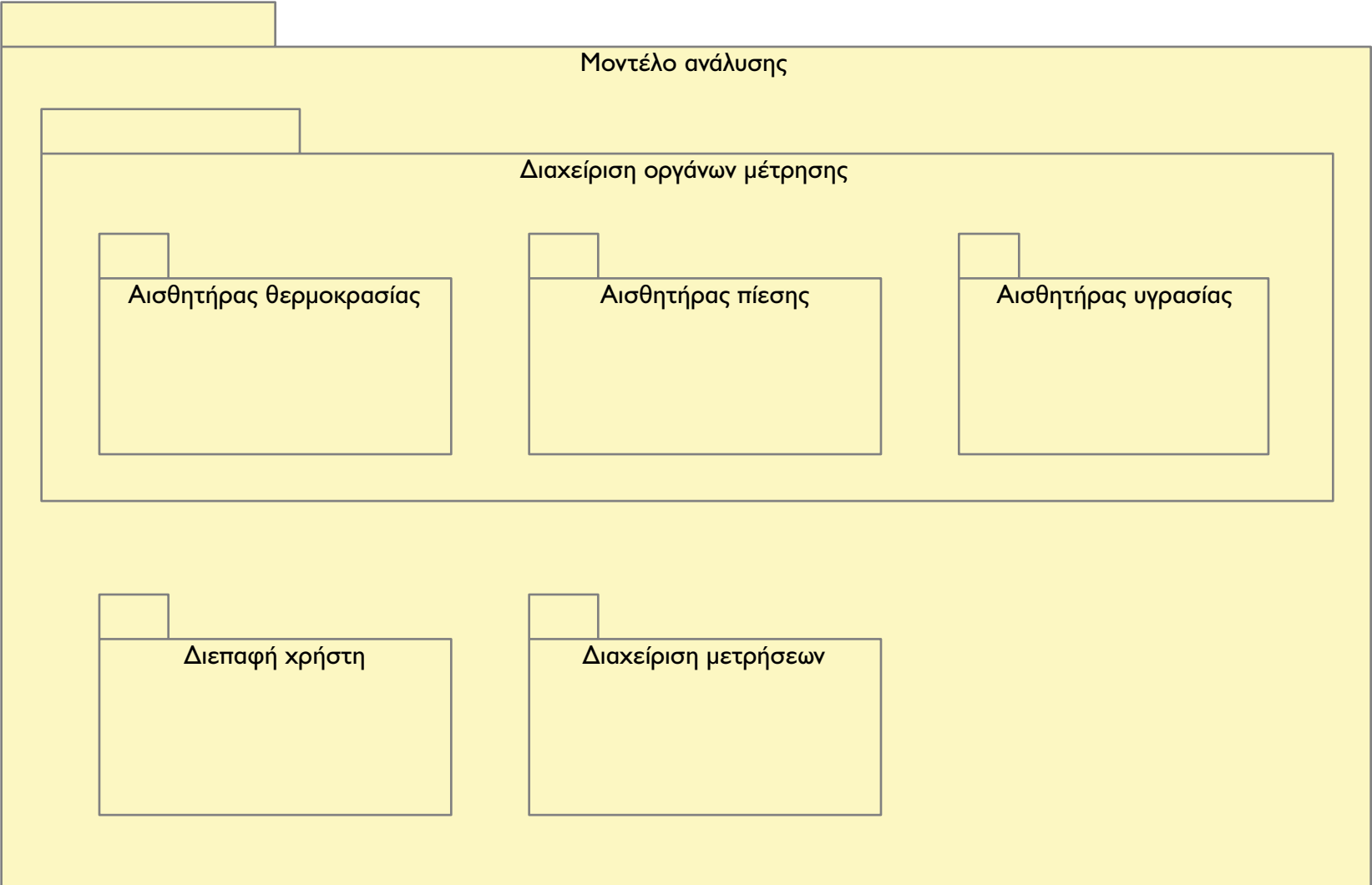
Στο σημείο αυτό, εκτός από τον ορισμό των πακέτων ανάλυσης, μπορεί να γίνει και ένας πρώτος εντοπισμός κάποιων κλάσεων οντοτήτων, συνήθως των πιο προφανών, οι οποίες προκύπτουν αβίαστα από την αντίληψη για το πρόβλημα. Η κύρια δουλειά εντοπισμού των κλάσεων ανάλυσης θα γίνει, πάντως, κατά το επόμενο βήμα, αυτό της ανάλυσης περιπτώσεων χρήσης.

Δεν έχει νόημα η συζήτηση για τον καλύτερο ορισμό πακέτων ανάλυσης, ειδικά όταν αυτή γίνεται σε αφηρημένο επίπεδο, χωρίς να υπάρχει κάποιο συγκεκριμένο πρόβλημα όπου να αναφέρεται. Η εμπειρία, η δημιουργικότητα και η αντίληψη του κατασκευαστή πρέπει, εφαρμοζόμενες στις εκάστοτε συνθήκες πεδίου προβλήματος (application domain) και ανάπτυξης λογισμικού (χαρακτηριστικά ομάδας και περιβάλλοντος ανάπτυξης), να τον οδηγήσουν στον ορισμό της αρχικής δομής του μοντέλου ανάλυσης, η οποία μπορεί να βελτιώνεται καθώς η ανάπτυξη προχωρά.

## **Παράδειγμα 2/Κεφάλαιο 8**

Στο Σχήμα 8.16 δίνεται ένας τρόπος ορισμού των πακέτων ανάλυσης για το πρόβλημα που περιγράφεται στο Παράδειγμα Ι του κεφαλαίου αυτού.

**Σχήμα 8.16** Πακέτα ανάλυσης για το παράδειγμα Ι (Σχήμα 8.5).



Στο πρώτο επίπεδο διακρίνουμε τρία πακέτα. Ένα είναι υπεύθυνο για τη διαχείριση των οργάνων (αισθητήρων) μέτρησης, ένα για την επικοινωνία με το χρήστη και ένα για τη διαχείριση των μετρήσεων.

Σχετικά με την ανάγκη ύπαρξης του πρώτου, αυτή επιβάλλεται από την απαίτηση οποιαδήποτε μεταβολή στη συμπεριφορά των εξωτερικών συσκευών να μην επηρεάζει ολόκληρη την εφαρμογή λογισμικού. Οπότε, καλώς ορίζεται το πακέτο, έτσι ώστε οι ενδεχόμενες μεταβολές να περιοριστούν σε αυτό. Με το ίδιο σκεπτικό αιτιολογείται και το δεύτερο επίπεδο ορισμού πακέτων ανάλυσης, ένα για κάθε αισθητήρα.

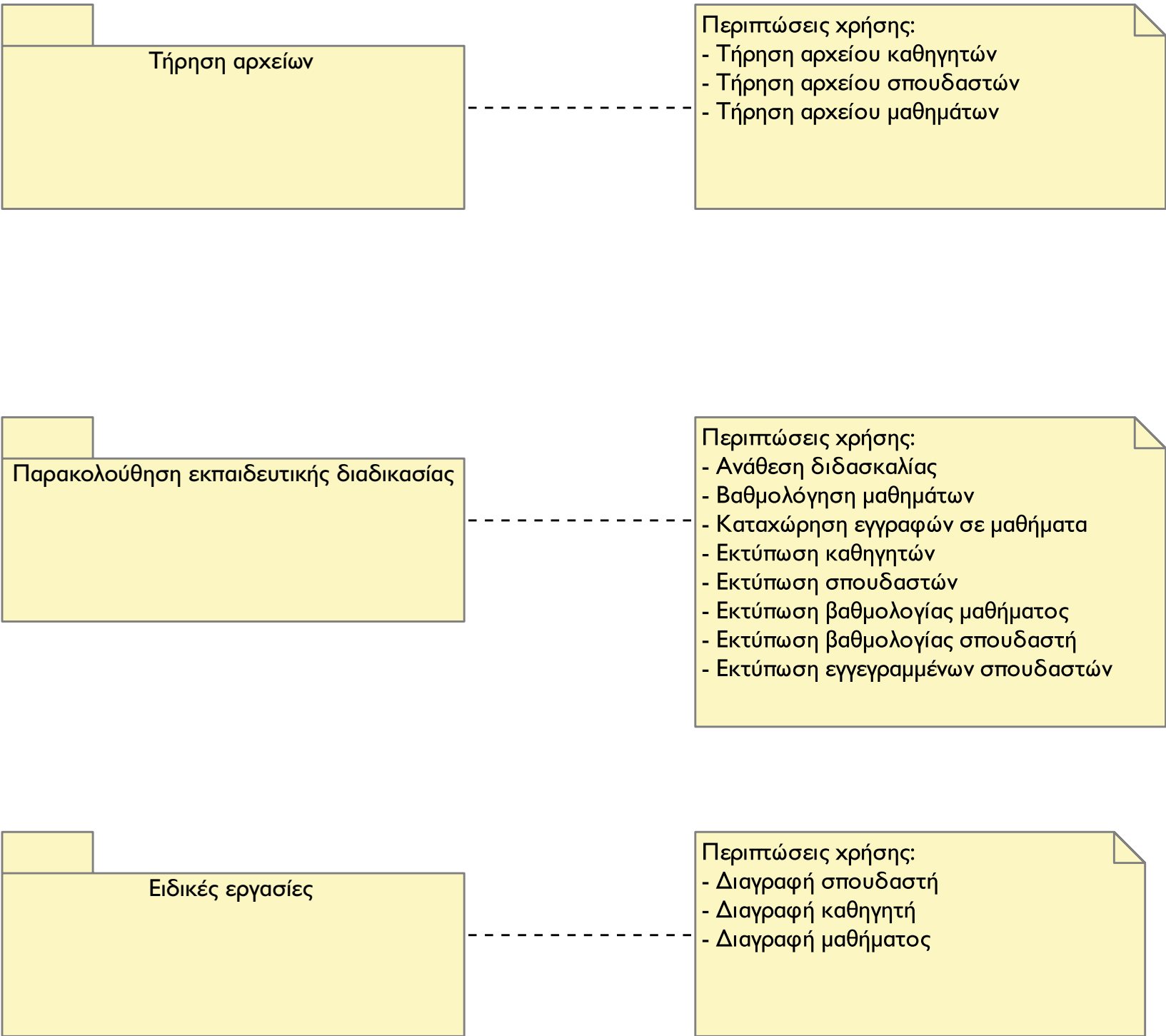
Η διάκριση της επικοινωνίας χρήστη από τη διαχείριση των μετρήσεων δεν είναι απαραίτητη, εκτός αν θέλουμε να χρησιμοποιούμε εναλλακτικές εικονικές (visual) μορφές των οργάνων μέτρησης ή, γενικά, να έχουμε τη δυνατότητα να αναδομήσουμε τη διεπαφή χρήστη χωρίς επιπτώσεις στην υπολογιστική εργασία του λογισμικού.

### **Μελέτη περίπτωσης**

Εφαρμόζοντας τα αναφερόμενα στην Ενότητα 8.5.1, καταλήγουμε στη δική μας εκδοχή σχετικά με τα πακέτα ανάλυσης της εφαρμογής λογισμικού «Επίκουρος». Αυτή δίνεται στο Σχήμα 8.17.



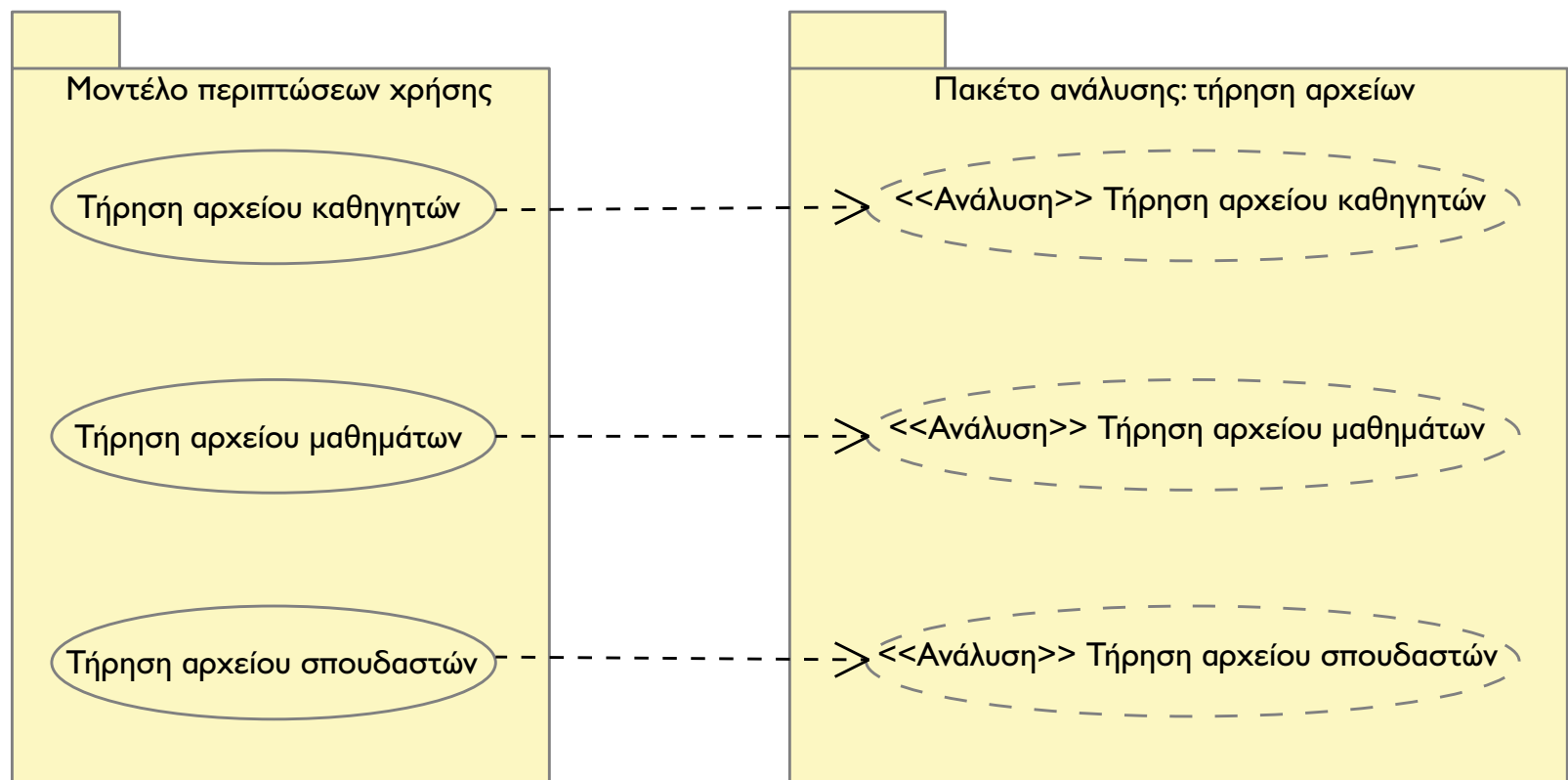
**Σχήμα 8.17** Δομή του μοντέλου ανάλυσης της εφαρμογής «Επίκουρος».



Πρόκειται για μια δομή ενός επιπέδου, όπου περιλαμβάνονται τρία πακέτα ανάλυσης. Το πρώτο θα περιλαμβάνει τις κλάσεις που σχετίζονται με τις εργασίες διαχείρισης αρχείων, το δεύτερο εκείνες που σχετίζονται με το καθημερινό τρέξιμο του συστήματος, ενώ το τρίτο περιλαμβάνει κλάσεις που σχετίζονται με την πραγματοποίηση ειδικών εργασιών που δεν συμβαίνουν σε καθημερινή βάση.

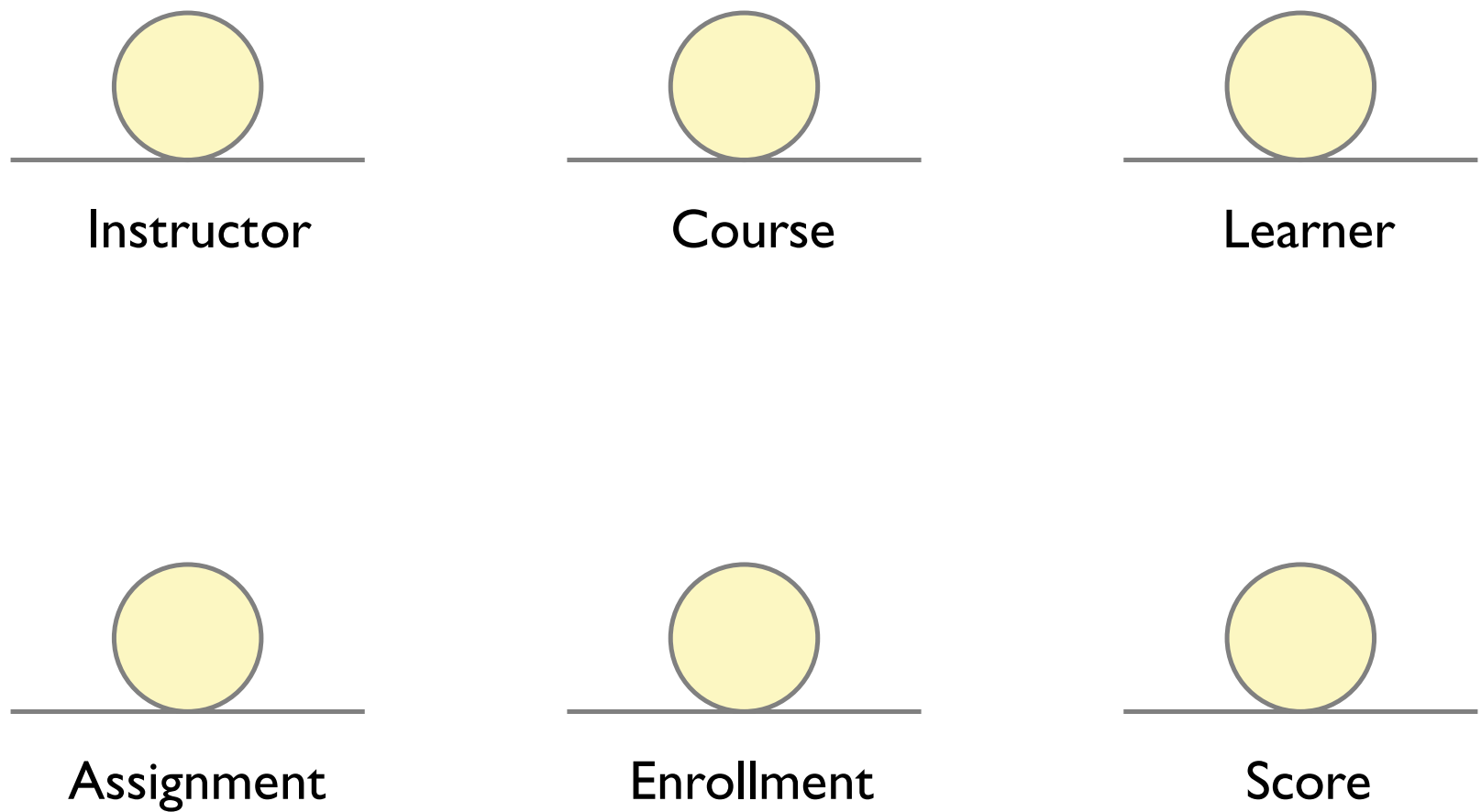
Στο Σχήμα 8.18 διακρίνεται η αντιστοίχιση των περιπτώσεων χρήσης που φαίνονται στο Σχήμα 8.17 σε περιπτώσεις χρήσης πεδίου ανάλυσης για το πρώτο πακέτο ανάλυσης. Ανάλογη είναι η κατάσταση και για τα άλλα δύο πακέτα ανάλυσης.

**Σχήμα 8.18** Αντιστοίχιση περιπτώσεων χρήσης σε περιπτώσεις χρήσης πεδίου ανάλυσης για ένα από τα πακέτα ανάλυσης της εφαρμογής «Επίκουρος».



Ακολουθώντας, θα προχωρήσουμε στον εντοπισμό των κλάσεων οι οποίες –ήδη από το σημείο αυτό– είναι φανερό ότι θα ανήκουν στο μοντέλο ανάλυσης. Οι κλάσεις αυτές είναι κλάσεις οντοτήτων του πεδίου του προβλήματος και φαίνονται στο Σχήμα 8.19.

**Σχήμα 8.19** Κλάσεις που είναι εύκολο να εντοπιστούν κατά την αρχιτεκτονική ανάλυση για το λογισμικό «Επίκουρος».



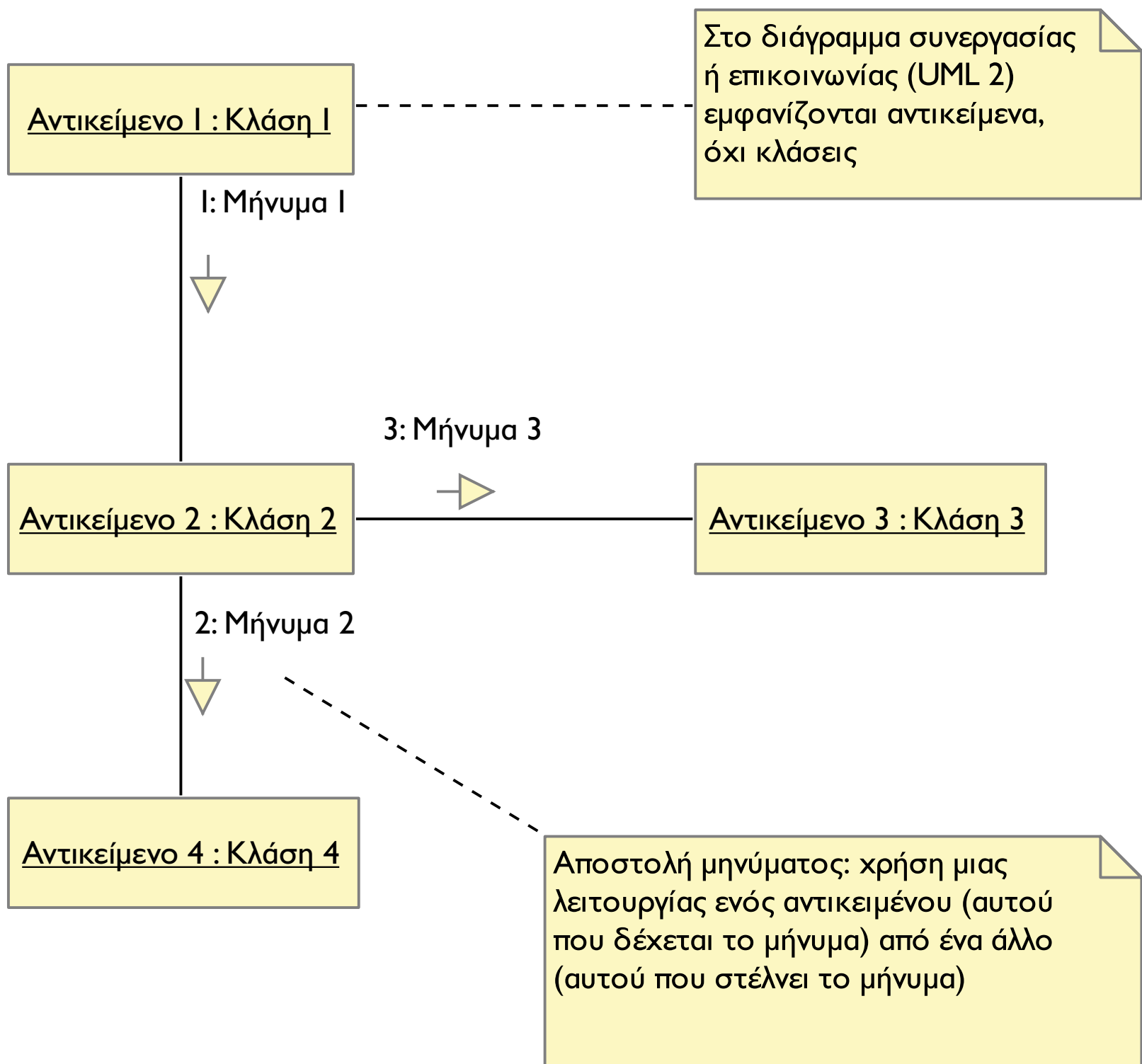
## Δραστηριότητα 7/Κεφάλαιο 8

Κατασκευάστε τα αντίστοιχα με το Σχήμα 8.18 διαγράμματα για τα άλλα δύο πακέτα ανάλυσης που φαίνονται στο Σχήμα 8.17.

### Συμβολισμοί UML

Ένα χρήσιμο εργαλείο για την περιγραφή της συνεργασίας μεταξύ κλάσεων σε μια περίπτωση χρήσης είναι το διάγραμμα συνεργασίας (collaboration diagram) της UML. Το διάγραμμα αυτό περιέχει τα αντικείμενα που συμμετέχουν σε μια συνεργασία, τα μηνύματα τα οποία ανταλλάσσονται μεταξύ τους, καθώς και τη σειρά με την οποία αυτό συμβαίνει. Στο Σχήμα 8.20 φαίνονται οι συμβολισμοί της UML για τα διαγράμματα συνεργασίας. Στη UML 2 χρησιμοποιείται και ο όρος «Διάγραμμα επικοινωνίας» (communication diagram).

**Σχήμα 8.20** Συμβολισμοί UML για τα διαγράμματα συνεργασίας ή επικοινωνίας.



### 8.5.2. Ανάλυση περιπτώσεων χρήσης

Στην αντικειμενοστρεφή τεχνολογία δομικές μονάδες λογισμικού είναι οι κλάσεις, τα αντικείμενα των οποίων παράγουν κατά την εκτέλεση του προγράμματος το επιθυμητό αποτέλεσμα. Η ανάλυση των περιπτώσεων χρήσης στοχεύει στον αρχικό καθορισμό των κλάσεων, καθώς και στην αλληλεπίδραση μεταξύ αυτών. Οι περιπτώσεις χρήσης αντιστοιχίζονται σε περιπτώσεις χρήσης πεδίου ανάλυσης. Για καθεμία από αυτές εντοπίζονται κλάσεις, αντικείμενα των οποίων συνεργάζονται μεταξύ τους ώστε να επιτύχουν το επιθυμητό αποτέλεσμα.

#### Εντοπισμός των κλάσεων

Όπως στην περίπτωση της αρχιτεκτονικής ανάλυσης, δεν υπάρχει μοναδικά ερμηνεύσιμος τρόπος για τον καθορισμό των κλάσεων. Οι κατευθυντήριες γραμμές που δίνει η ενοποιημένη προσέγγιση ως βήματα για τον καθορισμό των κλάσεων είναι οι ακόλουθες:

1. Μελετώντας την περιγραφή της περίπτωσης χρήσης εντοπίζονται τα δεδομένα που σχετίζονται με αυτή. Αυτά αντιστοιχούν σε κλάσεις οντοτήτων του πεδίου της ανάλυσης ή σε πεδία κατάστασης (fields) τέτοιων κλάσεων. Τι από τα δύο ισχύει ενδεχομένως να μην είναι δυνατόν να αποφασιστεί από την αρχή αλλά σε επόμενο κύκλο ανάπτυξης.
2. Για καθεμία από τις κλάσεις που εντοπίστηκαν στο προηγούμενο βήμα ορίζεται μια συνοριακή κλάση (interface) μέσω της οποίας γίνεται η προσπέλαση των υπηρεσιών της κατά την εκτέλεση του προγράμματος.
3. Για κάθε χειριστή ορίζεται μία τουλάχιστον συνοριακή κλάση η οποία αντιστοιχεί στη διεπαφή μέσω της οποίας ο χειριστής αλληλεπιδρά με το σύστημα. Στην περίπτωση που ο χειριστής αντιστοιχεί σε χρήστη – φυσικό πρόσωπο, τέτοιες κλάσεις τελικά απεικονίζονται σε φόρμες και στοιχεία της διεπαφής με το χρήστη (user interface) και πρέπει να οριστούν όσες τέτοιες κλάσεις χρειάζονται για να πραγματοποιηθεί η ροή της περίπτωσης χρήσης. Αν ο χειριστής αντιστοιχεί σε εξωτερικό σύστημα, οι κλάσεις αυτές



απεικονίζονται στις μονάδες ελέγχου της επικοινωνίας με αυτό, οι οποίες μπορούν να αντιστοιχούν σε πρωτόκολλα, οδηγούς συσκευών κ.ά.

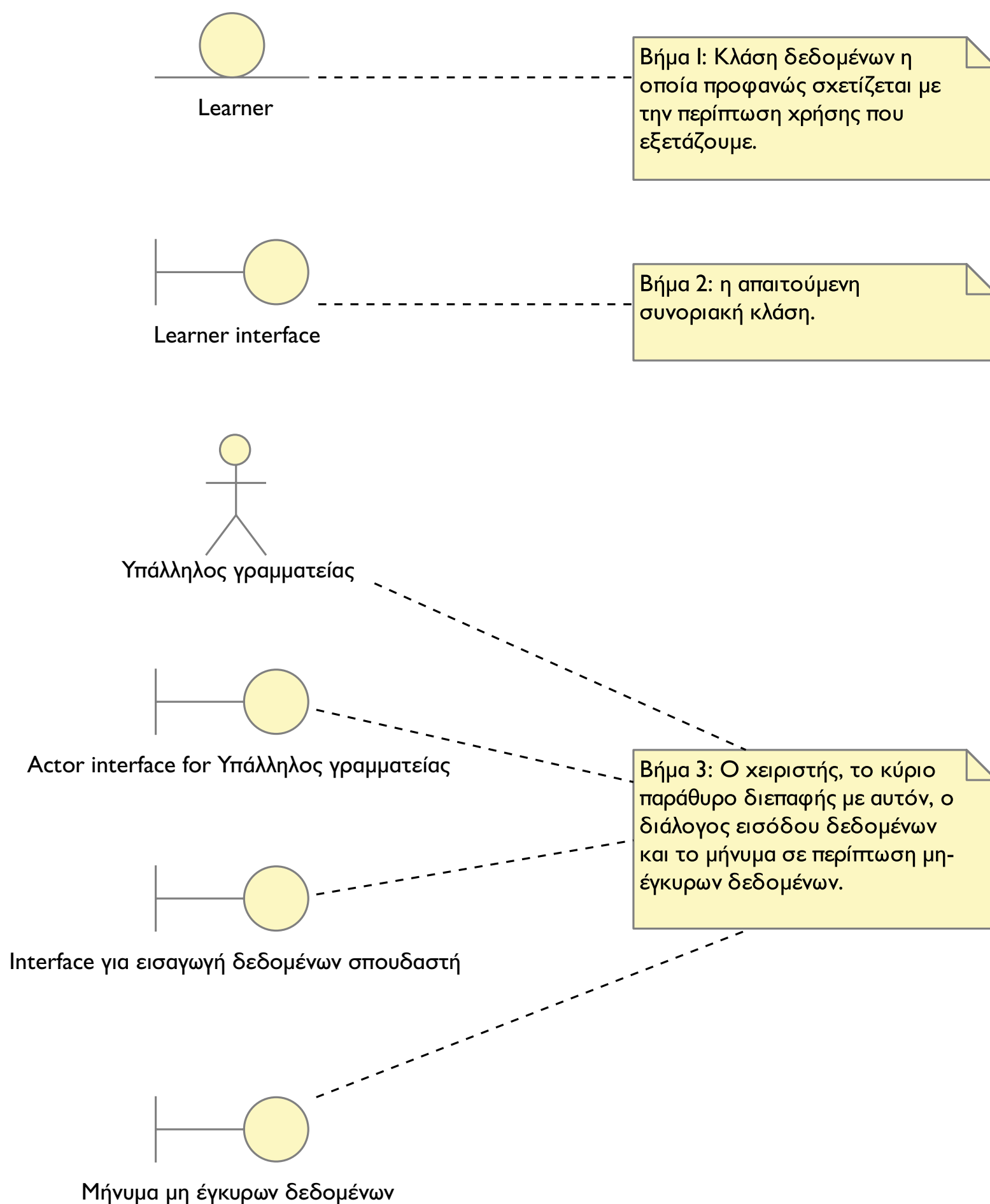
4. Ορίζεται μία τουλάχιστον κλάση ελέγχου για τον έλεγχο της ροής της περίπτωσης χρήσης. Σε απλές περιπτώσεις χρήσης, η κλάση αυτή μπορεί να ενσωματώνεται στην κλάση που αντιστοιχεί στη διεπαφή με τον χειριστή. Σε περισσότερο σύνθετες, αυτή μπορεί να αντιστοιχεί σε συστήματα διαχείρισης δοσοληψιών (transaction processing), μηνυμάτων κ.ά.

Όπως και με την αρχιτεκτονική ανάλυση, δεν υπάρχει μοναδικός τρόπος εκτέλεσης των παραπάνω εργασιών. Η δημιουργική αντίληψη, η εμπειρία αλλά και η διαίσθηση είναι σύμμαχοι της γνώσης που πρέπει να διαθέτει ο κατασκευαστής. Η αντιμετώπιση του προβλήματος σε μικρά τμήματα και με επαναλήψεις, όπως περιγράφεται από την ενοποιημένη προσέγγιση, μειώνει τις επιπτώσεις μιας αρχικής αστοχίας στον ορισμό των κλάσεων. Οι κλάσεις που εντοπίζονται στο σημείο αυτό αφορούν την υλοποίηση κάθε περίπτωσης χρήσης πεδίου ανάλυσης και καταγράφονται σε ένα διάγραμμα κλάσεων. Το διάγραμμα αυτό θα εμπλουτίζεται συνεχώς, ώστε να περιέχει, μετά την ανάλυση των κλάσεων, τις συσχετίσεις μεταξύ των κλάσεων και τις ιδιότητες των συσχετίσεων αυτών.

## Μελέτη περίπτωσης

Θα συνεχίσουμε τη μελέτη περίπτωσης εντοπίζοντας τις κλάσεις που προκύπτουν από την εφαρμογή των παραπάνω βημάτων στην περίπτωση χρήσης «τήρηση αρχείου σπουδαστών» με την οποία ασχοληθήκαμε στις αναφορές μας στην εφαρμογή «Επίκουρος» που προηγήθηκαν. Οι κλάσεις που εντοπίζονται, μαζί με κάποια χρήσιμα σχόλια, φαίνονται στο Σχήμα 8.21.

**Σχήμα 8.2I** Ανάλυση της περίπτωσης χρήσης «τήρηση αρχείου σπουδαστώ» της εφαρμογής «Επίκουρος» (I).



Το μόνο άξιο παρατήρησης είναι ότι, ενδεχομένως, δεν έχουμε ακόμη συλλάβει τον ρόλο της κλάσης ελέγχου που ορίσαμε. Σύμφωνα με τα αναφερόμενα παραπάνω, στο βήμα 4 είναι πιθανόν η κλάση αυτή να μην είναι απαραίτητη. Ωστόσο, σε αυτή τη φάση την ορίζουμε και θα διαπιστώσουμε στη συνέχεια αν είναι αναγκαία ή όχι.

### Δραστηριότητα 8/Κεφάλαιο 8

Εφαρμόζοντας τα βήματα που περιγράφονται στην Ενότητα 8.5.2 (εντοπισμός των κλάσεων), εντοπίστε τις κλάσεις από την ανάλυση της περίπτωσης χρήσης «διαγραφή σπουδαστή». Θα καταλήξετε κατασκευάζοντας ένα διάγραμμα κλάσεων παρόμοιο με αυτό που φαίνεται στο Σχήμα 8.21.

Τα πράγματα εδώ είναι λίγο πιο σύνθετα. Θα πρέπει να θυμηθείτε ότι, όπως προκύπτει από τον ορισμό των απαιτήσεων (απαίτηση 7), η διαγραφή σπουδαστή επιτρέπεται μόνο αν δεν έχει εγγραφεί σε κανένα μάθημα. Αρα, σχετιζόμενη κλάση οντοτήτων είναι και αυτή των εγγραφών. Επίσης, θα πρέπει να προσέξετε ότι απαιτούνται περισσότερα από ένα παράθυρα διαλόγου επικοινωνίας με το χειριστή, ώστε να εμφανίζονται τα μηνύματα προς αυτόν, όπως αυτό περιγράφεται τόσο στην προδιαγραφή της περίπτωσης χρήσης που δώσαμε ως μελέτη περίπτωσης όσο και στο αντίστοιχο διάγραμμα δραστηριοτήτων με το οποίο (πρέπει να) έχετε ασχοληθεί στη Δραστηριότητα 5.

### Εντοπισμός της συνεργασίας

Μετά τον ορισμό των κλάσεων, κατά την ανάλυση περιπτώσεων χρήσης, πρέπει να προσδιοριστεί η αλληλεπίδραση των αντικειμένων των κλάσεων αυτών κατά το τρέξιμο των περιπτώσεων χρήσης. Για τον σκοπό αυτό χρησιμοποιούνται διαγράμματα συνεργασίας. Ένα διάγραμμα συνεργασίας είναι μια περιγραφή της ροής γεγονότων μιας περίπτωσης χρήσης (που μέχρι το

σημείο αυτό έχει περιγραφεί από την οπτική του πελάτη με κείμενο ή και με διάγραμμα δραστηριότητας) από την οπτική γωνία του κατασκευαστή. Για την ακρίβεια, καλό είναι να κατασκευάζεται ένα διάγραμμα συνεργασίας για καθεμία από τις εναλλακτικές ροές της υπό ανάλυση περίπτωσης χρήσης.

Ξεκινώντας από την αρχή της ροής γεγονότων και έχοντας υπόψη την περιγραφή της περίπτωσης χρήσης με κείμενο (Σχήμα 8.6) και διαγράμματα δραστηριοτήτων (Σχήμα 8.8), ακολουθούμε τα βήματά της και αποφασίζουμε ποια αντικείμενα των κλάσεων ανάλυσης που έχουν εντοπιστεί είναι απαραίτητα για την πραγματοποίηση κάθε βήματος. Αυτά αποτυπώνονται σε ένα διάγραμμα συνεργασίας (ή σε περισσότερα, αν υπάρχουν εναλλακτικές ροές). Μερικές κατευθυντήριες γραμμές σχετικά με την εργασία αυτή είναι οι ακόλουθες:

1. Το πρώτο μήνυμα αποστέλλεται πάντα από έναν χειριστή που εκκινεί την περίπτωση χρήσης προς ένα αντικείμενο συνοριακής κλάσης.
2. Καθεμία από τις κλάσεις ανάλυσης που έχουν προσδιοριστεί έχει τουλάχιστον ένα αντικείμενο που συμμετέχει στο διάγραμμα συνεργασίας. Αν διαπιστωθεί ότι η ροή υλοποιείται και περισσεύουν κλάσεις, τότε αυτές μάλλον δεν θα έπρεπε να έχουν οριστεί.
3. Τόσο τα αντικείμενα όσο και τα μηνύματα δεν περιγράφονται στο σημείο αυτό με αυστηρότητα και λεπτομέρεια, αλλά με σκοπό να επιδείξουν τα ποιοτικά χαρακτηριστικά της υλοποίησης της περίπτωσης χρήσης.
4. Η αποστολή μηνύματος από ένα αντικείμενο σε ένα άλλο είναι πιθανό ότι υποδηλώνει συσχέτιση μεταξύ των αντίστοιχων κλάσεων, η οποία θα αποκαλυφθεί ως τέτοια αργότερα, κατά την ανάλυση κλάσεων.
5. Η αναγραφή της αλληλουχίας της συνεργασίας (σειρά αποστολής μηνυμάτων) σε αυτήν τη φάση δεν είναι υποχρεωτική και μπορεί να παραλείπεται.

Τόσο από τον εντοπισμό των κλάσεων όσο και από τον εντοπισμό της συνεργασίας αναδεικνύεται η μεγάλη σημασία των περιπτώσεων χρήσης και

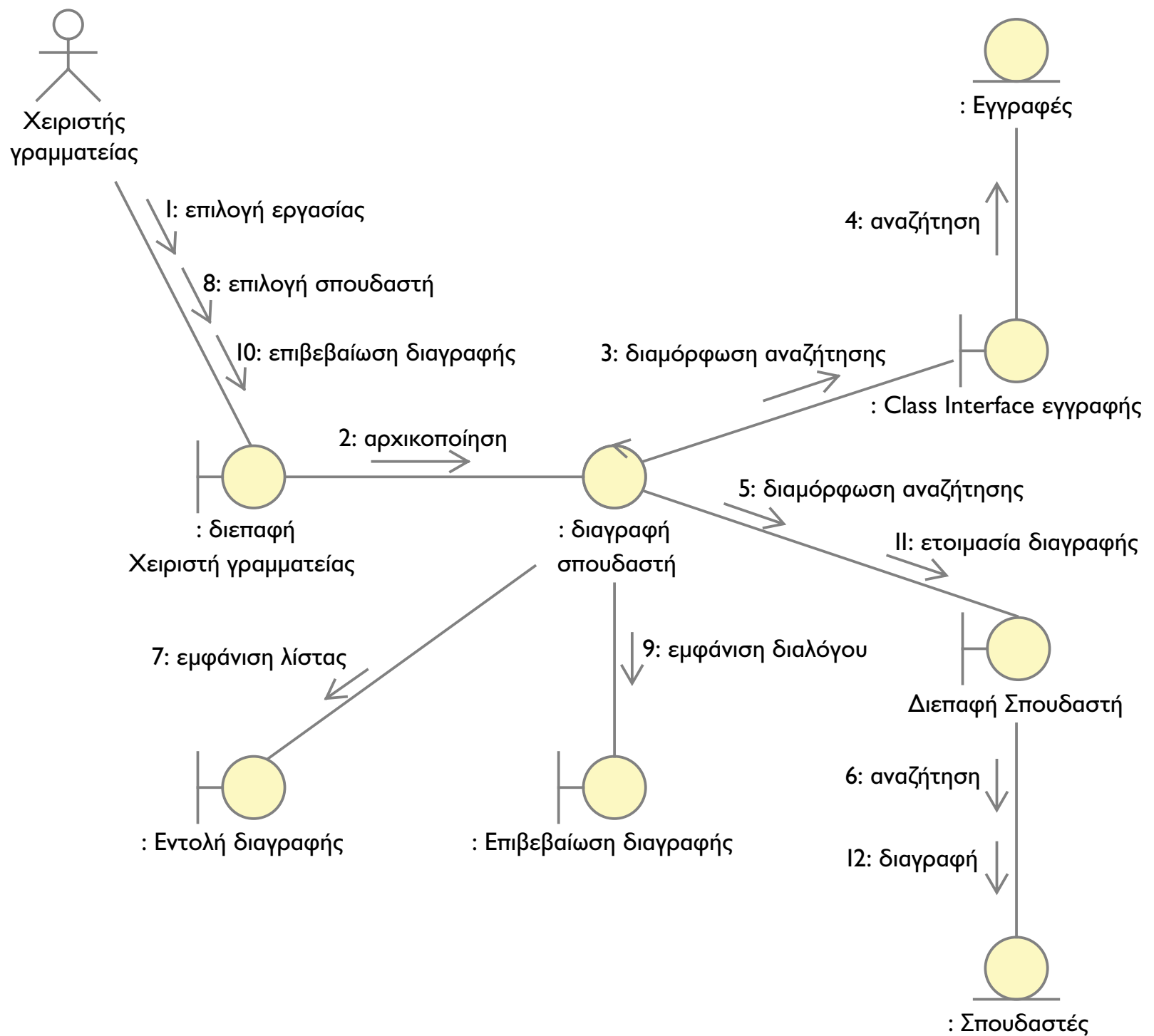
της τεκμηρίωσής τους στην αντικειμενοστρεφή ανάπτυξη λογισμικού με την ενοποιημένη προσέγγιση. Κάθε κλάση ανάλυσης ανήκει σε ένα πακέτο ανάλυσης, το οποίο σχετίζεται με μία περίπτωση χρήσης πεδίου ανάλυσης, η οποία, με τη σειρά της, αντιστοιχεί σε μία από τις περιπτώσεις χρήσης που ορίστηκαν κατά τον καθορισμό των απαιτήσεων. Ξεκινώντας από οποιαδήποτε κλάση, είναι δυνατόν να εντοπίσουμε μία τουλάχιστον περίπτωση χρήσης στην οποία αντιστοιχεί, δηλαδή μία λειτουργική απαίτηση την οποία αυτή ικανοποιεί.

### **Μελέτη περίπτωσης**

Θα κατασκευάσουμε τα διαγράμματα συνεργασίας για την περίπτωση χρήσης πεδίου ανάλυσης «διαγραφή σπουδαστή», η οποία είναι πιο σύνθετη από την «τήρηση αρχείου σπουδαστών». Θα πρέπει να χρησιμοποιήσουμε όλες τις κλάσεις που προσδιορίσαμε κατά τον προσδιορισμό των κλάσεων γι' αυτή την περίπτωση χρήσης (Δραστηριότητα 7, Σχήμα 8.32) και να ορίσουμε συνδέσμους συνεργασίας μεταξύ τους, καθώς και ανταλλασσόμενα μηνύματα, ώστε να ικανοποιείται η ροή εργασιών της προδιαγραφής της περίπτωσης χρήσης (Σχήμα 8.29 και Σχήμα 8.30).

Στο Σχήμα 8.22 φαίνεται το διάγραμμα συνεργασίας που σχετίζεται με την κύρια ροή, ενώ στο Σχήμα 8.23 φαίνεται το αντίστοιχο διάγραμμα για τη δευτερεύουσα ροή.

**Σχήμα 8.22** Διάγραμμα συνεργασίας για την κύρια ροή της περίπτωσης χρήσης «διαγραφή σπουδαστή».



Με την επιλογή εργασίας από το χρήστη (1) αρχικοποιείται η μονάδα ελέγχου της περίπτωσης χρήσης (2). Μετά την αρχικοποίηση, τρέχει μια ερώτηση προς το αρχείο εγγραφών σπουδαστών σε μαθήματα από την οποία θα προκύψει η λίστα των σπουδαστών (Σ1) που έχουν εγγραφεί σε μαθήματα (3, 4). Η λίστα αυτή συγκρίνεται με εκείνη (Σ2) που προκύπτει από την ερώτηση προς το αρχείο σπουδαστών (5, 6).

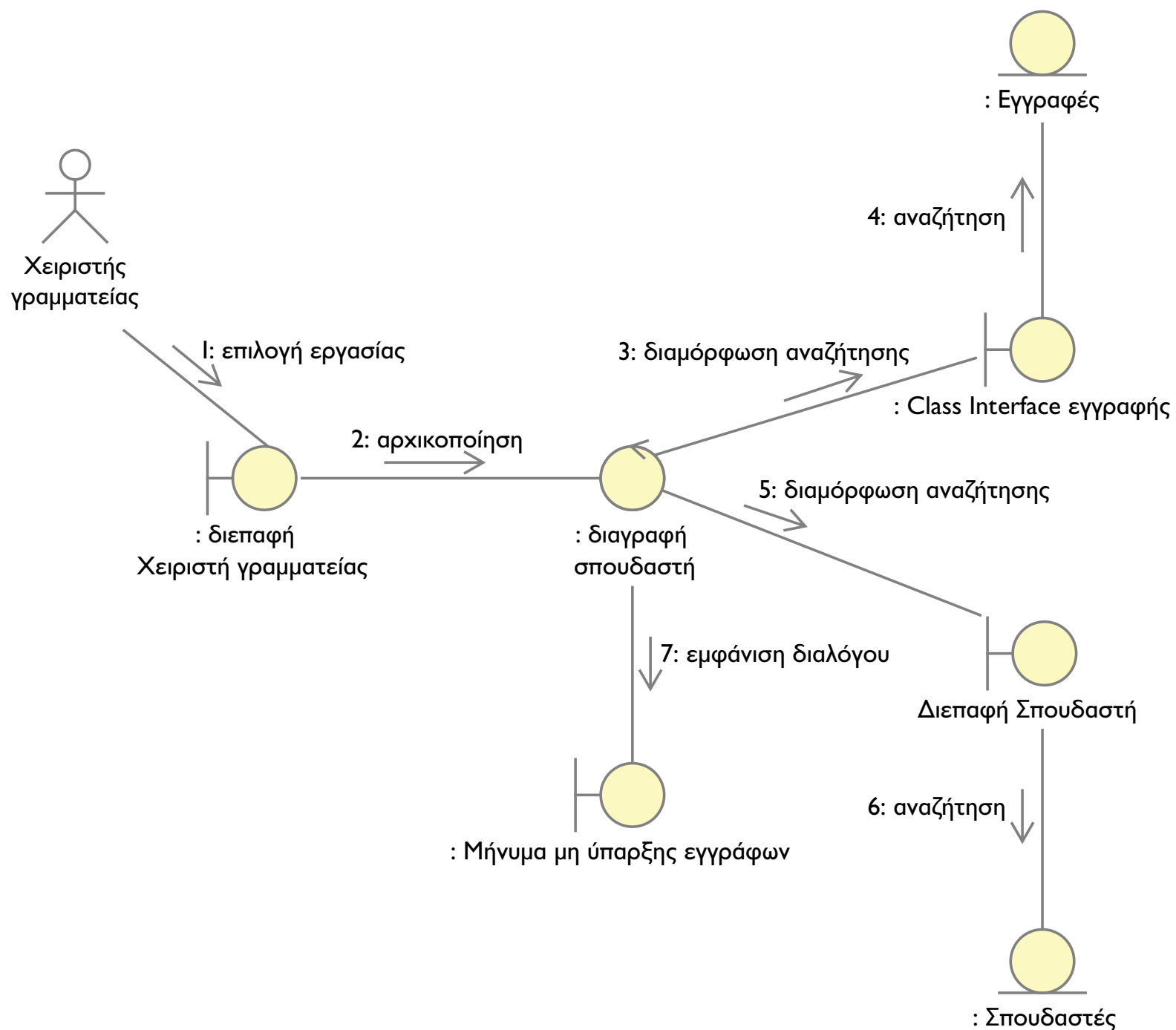
Αν διαπιστωθεί ότι υπάρχουν εγγραφές της Σ2 που δεν ανήκουν στη Σ1, η εργασία προχωρεί (αυτό συμβαίνει στην κύρια ροή) και εμφανίζεται ο διάλογος με τις εγγραφές που επιτρέπεται να διαγραφούν, δηλαδή τις Σ2-Σ1 (7). Ο χρήστης επιλέγει έναν σπουδαστή και δίνει την εντολή διαγραφής (8). Ακολούθως εμφανίζεται διάλογος επιβεβαίωσης της διαγραφής (9), το ερώτημα του οποίου επιβεβαιώνει ο χρήστης (10). Τέλος, ετοιμάζεται και εκτελείται η διαγραφή από το αρχείο σπουδαστών (11, 12).

Μπορεί κανείς εύκολα να διαπιστώσει ότι αυτή η περιγραφή ανταποκρίνεται στη ροή εργασιών της περίπτωσης χρήσης που περιγράφηκε προηγουμένως. Τα βήματα τα οποία είναι ίσως λιγότερο κατανοητά είναι αυτά που αναφέρονται στα συνοριακά αντικείμενα των αντικειμένων που αντιστοιχούν σε κλάσεις οντοτήτων. Το θέμα αυτό όμως δεν αφορά (επαναλαμβάνουμε) την ανάλυση. Αρκεί που εντοπίστηκε μια συνεργασία που περιλαμβάνει αντικείμενα όλων των κλάσεων ανάλυσης. Κατά τη σχεδίαση, οι συνθήκες θα υποδείξουν ποια από αυτά τα αντικείμενα θα παραμείνουν και ποια όχι.

Μια άλλη παρατήρηση αφορά το ρόλο της κλάσης ελέγχου που ορίσαμε για την περίπτωση χρήσης. Αυτή είναι πιθανό να ενσωματωθεί στη συνέχεια σε ένα από τα παράθυρα διαλόγου με τον χειριστή. Ούτε αυτό όμως το θέμα απασχολεί την ανάλυση.



**Σχήμα 8.23** Διάγραμμα συνεργασίας για τη δευτερεύουσα ροή της περίπτωσης χρήσης «διαγραφή σπουδαστή».





## Δραστηριότητα 9/Κεφάλαιο 8

Σχολιάστε το διάγραμμα συνεργασίας για τη δευτερεύουσα ροή της περίπτωσης χρήσης «διαγραφή σπουδαστή» (Σχήμα 8.23).

## Δραστηριότητα 10/Κεφάλαιο 8

Προσπαθήστε να κατασκευάσετε τα διαγράμματα συνεργασίας για την περίπτωση χρήσης «τήρηση αρχείου σπουδαστών». Έχοντας προσέξει τη μελέτη περίπτωσης είναι εύκολο, δεν συμφωνείτε;

### 8.5.3. Ανάλυση κλάσεων

Η εργασία που έπεται του εντοπισμού των κλάσεων είναι η ανάλυσή τους, έτσι ώστε να προσδιοριστούν οι ευθύνες τους, δηλαδή οι εργασίες που επιτελούν στις περιπτώσεις χρήσης με τις οποίες σχετίζονται, τα πεδία τους (fields), καθώς και οι συσχετίσεις τους με άλλες κλάσεις. Στη γενική περίπτωση, μια κλάση μπορεί να εμπλέκεται στην ανάλυση περισσότερων από μίας περιπτώσεων χρήσης. Η σύνθεση όλων των (ενδεχομένως) διαφορετικών εργασιών που μία κλάση επιτελεί στην ανάλυση των περιπτώσεων χρήσης αποτελεί το πεδίο ευθύνης της κλάσης αυτής.

#### Προσδιορισμός πεδίων

Τα πεδία αντιστοιχούν στην περιγραφή της κατάστασης κάθε κλάσης. Ο προσδιορισμός τους γίνεται με γνώμονα το «τι πρέπει να γνωρίζει» κάθε κλάση, ώστε να αντεπεξέλθει στις ευθύνες της. Μερικές χρήσιμες κατευθυντήριες γραμμές για τον καθορισμό των πεδίων είναι οι ακόλουθες:

- Κάθε πεδίο έχει ένα όνομα το οποίο είναι ουσιαστικό.
- Για την εργασία της ανάλυσης δεν ενδιαφέρουν οι τεχνικές λεπτομέρειες του πεδίου οι οποίες σχετίζονται με την υλοποίησή του. Αρκεί η ιδέα που υποδηλώνεται από το όνομά του.

- Τα σύνθετα πεδία (δηλαδή αυτά που τελικά θα είναι και τα ίδια κλάσεις) τα οποία ενδεχομένως να εντοπιστούν θα πρέπει να επαναχρησιμοποιούνται όσο το δυνατόν περισσότερο.
- Αν μια κλάση καταλήγει να έχει πολλά και σύνθετα πεδία, τα οποία κάνουν δύσκολη την αντίληψή της, θα πρέπει να εξετάζεται η δυνατότητα κατάτμησης της κλάσης σε περισσότερες.
- Τα πεδία των κλάσεων οντοτήτων είναι εύκολο να καθοριστούν από το πεδίο του προβλήματος. Αντιστοιχούν στις πληροφορίες τις οποίες το λογισμικό πρέπει να «γνωρίζει» για την αντίστοιχη οντότητα.
- Τα πεδία των συνοριακών κλάσεων που σχετίζονται με χειριστές που αντιστοιχούν σε χρήστες – φυσικά πρόσωπα είναι τα συστατικά της διεπαφής χρήστη (user interface), όπως ετικέτες, κουμπιά, θυρίδες εισόδου κειμένου, λίστες κύλισης κ.ά.
- Τα πεδία των συνοριακών κλάσεων που σχετίζονται με χειριστές – εξωτερικά συστήματα είναι συνήθως συστατικά κάποιας διεπαφής επικοινωνιών, πρωτοκόλλου, εγγραφής ανταλλαγής μηνυμάτων κ.λπ.
- Οι κλάσεις ελέγχου συνήθως δεν έχουν πεδία, διότι υπάρχουν για να παρέχουν υπηρεσίες και όχι για να αποθηκεύουν δεδομένα. Εξαίρεση μπορεί να αποτελούν περιπτώσεις προσωρινών μετρητών (counters) μεγεθών που σχετίζονται με τέτοιες κλάσεις.

## Προσδιορισμός σχέσεων

Όπως αναφέραμε, υπάρχουν τρία είδη σχέσεων μεταξύ κλάσεων: οι συσχετίσεις (associations), η σχέση συναρμολόγησης (aggregation) και η σχέση γενίκευσης (generalization). Στο σημείο αυτό εντοπίζονται για πρώτη φορά οι σχέσεις αυτές στο πεδίο της ανάλυσης.

Κατά τη δημιουργία των διαγραμμάτων συνεργασίας στην ανάλυση των περιπτώσεων χρήσης έχουν προσδιοριστεί σύνδεσμοι μεταξύ αντικειμένων, αναγκαίοι για να επιτευχθεί ο σκοπός κάθε περίπτωσης χρήσης. Οι σύνδεσμοι αυτοί ενδέχεται να αντιστοιχούν σε συσχετίσεις μεταξύ των αντίστοιχων κλάσεων. Συσχετίσεις επίσης εξάγονται από το θεματικό πεδίο του προβλήματος.

Οι σχέσεις συναρμολόγησης μεταξύ δύο αντικειμένων (και, κατ' επέκταση, μεταξύ των αντιστοίχων κλάσεων) αναγνωρίζονται σε τρεις περιπτώσεις: όταν το ένα περιέχει το άλλο (όπως για παράδειγμα ένα δοχείο περιέχει νερό), όταν το ένα αποτελείται από το άλλο (όπως για παράδειγμα ένα αυτοκίνητο αποτελείται από μηχανή και τροχούς), καθώς και όταν αυτά σχετίζονται με σημασιολογική σχέση του τύπου πατέρας – παιδί, διευθυντής – υπάλληλος κ.λπ. Όταν ισχύει κάτι από τα παραπάνω, είναι πιθανό (και όχι υποχρεωτικό) να υπάρχει σχέση συναρμολόγησης.

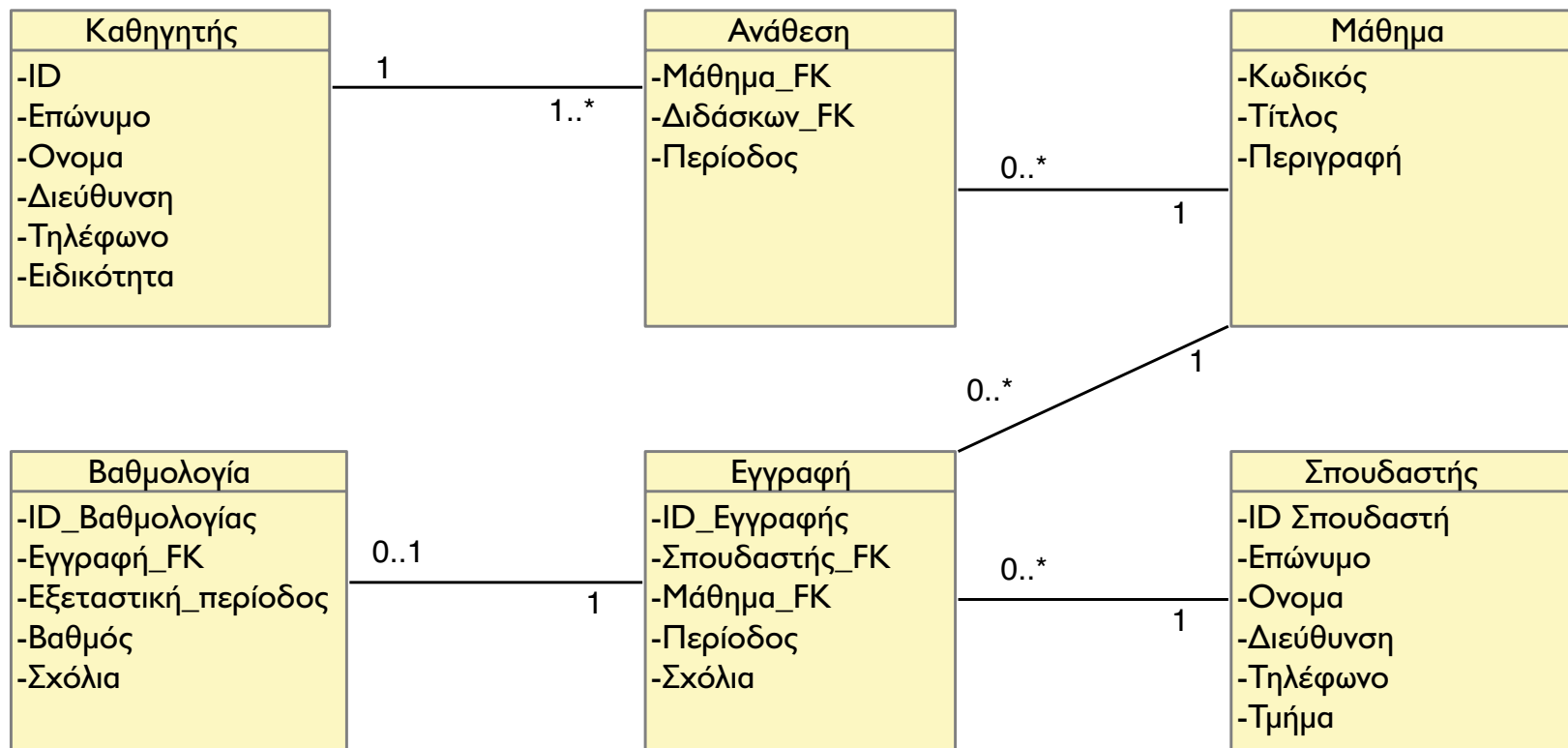
Τέλος, οι σχέσεις γενίκευσης στην ανάλυση ορίζονται προκειμένου να εξαχθεί ο «κοινός παράγοντας» της συμπεριφοράς δύο ή περισσότερων κλάσεων, ώστε να γίνει πιο κατανοητό το μοντέλο ανάλυσης. Η εξυπηρέτηση άλλων στόχων, όπως η κατασκευαστική ευκολία, η επαναχρησιμοποίηση κώδικα κ.λπ., δεν ενδιαφέρει κατά την εργασία της ανάλυσης.

Γενικά, κατά τον προσδιορισμό σχέσεων μεταξύ κλάσεων θα πρέπει να ισχύει το αρχαίο ρητό «το λακωνίζειν εστί φιλοσοφείν». Αυτό σημαίνει ότι πρέπει να ορίζονται όσο το δυνατόν λιγότερες σχέσεις, προκειμένου απλώς και μόνο να ικανοποιούνται οι απαιτήσεις των περιπτώσεων χρήσης. Κάθε υποψήφια σχέση θα πρέπει να αμφισβητείται τόσο πριν όσο και μετά τον ορισμό της, εξεταζόμενη υπό το πρίσμα του ερωτήματος «αν εκλείψει, θα τρέχουν οι περιπτώσεις χρήσης στις οποίες εμπλέκεται, δηλαδή θα ανταποκρίνεται στις ευθύνες της;». Αν η απάντηση στο ερώτημα αυτό είναι καταφατική, τότε η σχέση θα πρέπει να μην οριστεί.

## **Μελέτη περίπτωσης**

Με βάση όσα έχουμε αναφέρει σχετικά με τη μελέτη περίπτωσης μέχρι το σημείο αυτό αλλά και από τη γενικότερη ενασχόλησή μας με το πρόβλημα στο βιβλίο αυτό, είναι εύκολο να προσδιορίσουμε τα πεδία όλων των κλάσεων οντοτήτων που έχουμε εντοπίσει (Σχήμα 8.19). Επίσης, θα εντοπίσουμε τις συσχετίσεις μεταξύ των κλάσεων οντοτήτων, οι οποίες μπορούμε να θεωρούμε ότι αντιστοιχούν στις σχέσεις ενός σχεσιακού μοντέλου δεδομένων. Αυτά φαίνονται στο Σχήμα 8.24.

**Σχήμα 8.24** Πεδία και συσχετίσεις για τις κλάσεις οντοτήτων του λογισμικού «Επίκουρος».



#### 8.5.4. Ανάλυση πακέτων

Η τελευταία από τις εργασίες της αντικειμενοστρεφούς ανάλυσης, σύμφωνα με την ενοποιημένη προσέγγιση, μπορούμε να πούμε ότι «κλείνει τον κύκλο» και ασχολείται με τα πακέτα ανάλυσης τα οποία ορίστηκαν στην πρώτη από τις εργασίες της ανάλυσης. Σκοπός της είναι να επαληθεύσει την επίτευξη της μεγαλύτερης δυνατής ανεξαρτησίας μεταξύ των πακέτων, την ολοκλήρωση της ανάλυσης όλων των περιπτώσεων χρήσης, καθώς και να εντοπίσει τις συσχετίσεις μεταξύ πακέτων ανάλυσης. Όπως αναφέραμε και στην αρχή, δεν υπάρχει μοναδική συνταγή για να ορίζονται τα πακέτα ανάλυσης και πάντα θα υπάρχει αντίλογος.

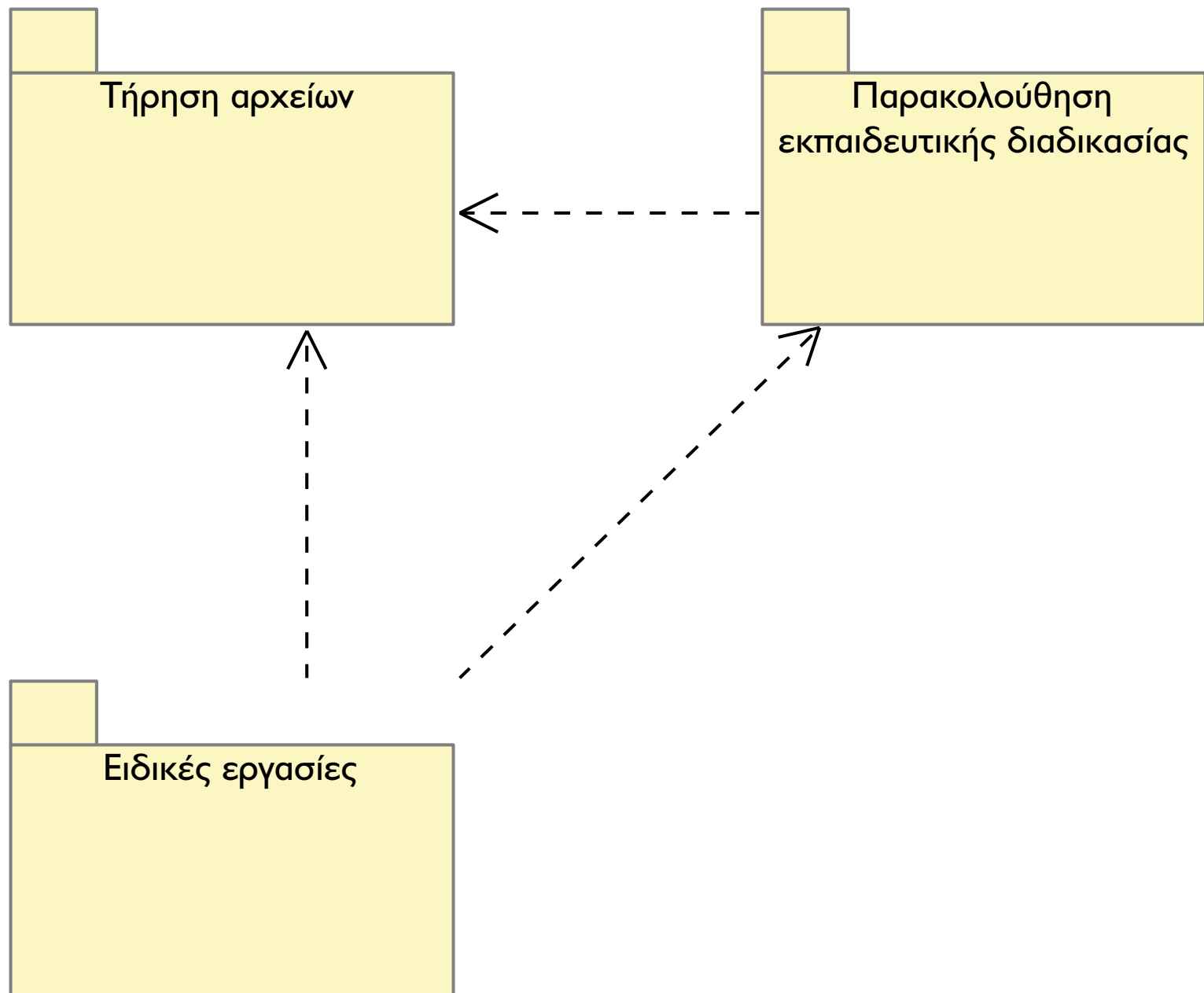
Μερικές γενικές κατευθυντήριες γραμμές για την εργασία αυτή είναι οι ακόλουθες:

- Κάθε πακέτο θα πρέπει να περιέχει σημασιολογικά συναφείς κλάσεις. Αν διαπιστωθεί ότι κάποιες από τις κλάσεις που περιέχει ταξινομούνται καλύτερα σε κάποιο άλλο πακέτο, τότε καλύτερα να μεταφερθούν εκεί.
- Αν μια κλάση ενός πακέτου A σχετίζεται με μια κλάση ενός πακέτου B, τότε θα πρέπει να ορίσουμε μια σχέση μεταξύ των πακέτων A και B.
- Οι σχέσεις μεταξύ πακέτων πρέπει να είναι οι λιγότερες δυνατές. Αν με τη μετακίνηση κλάσεων μεταξύ πακέτων επιτυγχάνουμε τόσο καλύτερη σημασιολογική συνάφεια όσο και ελαχιστοποίηση των σχέσεων μεταξύ πακέτων, τότε η μετακίνηση αυτή είναι σκόπιμη.

#### Μελέτη περίπτωσης

Αν εξαντλούσαμε την αναφορά μας στη μελέτη περίπτωσης αναλύοντας όλες τις περιπτώσεις χρήσης τις οποίες αποδώσαμε σε κάθε πακέτο ανάλυσης, θα προέκυπταν μεταξύ των πακέτων ανάλυσης οι σχέσεις που φαίνονται στο Σχήμα 8.25.

**Σχήμα 8.25** Σχέσεις μεταξύ πακέτων ανάλυσης για την εφαρμογή λογισμικού «Επίκουρος».



Και με βάση όσα έχουμε αναφέρει, οι σχέσεις αυτές μπορούν να εντοπιστούν συνδυάζοντας τις συσχετίσεις μεταξύ κλάσεων οντοτήτων (Σχήμα 8.24) και την απόδοση των περιπτώσεων χρήσης σε πακέτα ανάλυσης (Σχήμα 8.17).

## Σύνοψη ενότητας

---

Σύμφωνα με την ενοποιημένη προσέγγιση, η αντικειμενοστρεφής ανάλυση λογισμικού λαμβάνει χώρα σε τέσσερα βήματα. Κατά την αρχιτεκτονική ανάλυση καθορίζονται τα πακέτα του μοντέλου ανάλυσης τα οποία αντιστοιχούν στην κατάτμηση του προβλήματος σε περισσότερα του ενός μικρά τμήματα. Κατά την ανάλυση περιπτώσεων χρήσης γίνεται ο αρχικός καθορισμός των κλάσεων που θα αποτελέσουν το λογισμικό, καθώς και της συνεργασίας μεταξύ τους, προκειμένου να υλοποιούνται όσα απαιτούν οι περιπτώσεις χρήσης. Κατά την ανάλυση των κλάσεων γίνεται ο αρχικός καθορισμός των πεδίων και των μεθόδων κάθε κλάσης, καθώς και των συσχετίσεων μεταξύ τους. Τέλος, κατά την ανάλυση πακέτων καθορίζονται οι συσχετίσεις μεταξύ των πακέτων ανάλυσης και επαληθεύεται η κατάτμηση σε πακέτα, η οποία έχει πραγματοποιηθεί.

## ΕΝΟΤΗΤΑ 8.6. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ

### Άσκηση 1/Κεφάλαιο 8

---

Δεν χρειάζεται παρά ένα απλό ξεφύλλισμα του Κεφαλαίου 2 για να εντοπίσουμε ότι τα μοντέλα κύκλου ζωής λογισμικού που έχουν το ζητούμενο πλεονέκτημα είναι το μοντέλο πρωτοτυποποίησης και το σπειροειδές και, ασφαλώς, το γενικό μοντέλο κύκλου ζωής λογισμικού. Συγχαρητήρια σε όσους δεν χρειάστηκε να καταφύγουν στο ξεφύλλισμα.

### Άσκηση 2/Κεφάλαιο 8

---

Στο μοντέλο του καταρράκτη, οι εργασίες «απαιτήσεις», «ανάλυση» κ.λπ. ταυτίζονται με τις φάσεις του κύκλου ζωής και εκτελούνται για ολόκληρη την εφαρμογή λογισμικού και όχι για κάποιο τμήμα της. Αυτό δεν ισχύει στην ενοποιημένη προσέγγιση όπου, όπως φαίνεται στο Σχήμα 8.1, οι εργασίες



αυτές εκτελούνται επαναληπτικά μέσα στους κύκλους ανάπτυξης, καθένας εκ των οποίων αφορά ένα μικρό μέρος της εφαρμογής είτε καθορίζοντας νέες λειτουργίες είτε εξειδικεύοντας τις υπάρχουσες.

Αυτό, βέβαια, ενέχει τον κίνδυνο του υπερβολικού κατακερματισμού της εφαρμογής και της αποσπασματικής της αντίληψης, όταν η ανάπτυξη διαφορετικών τμημάτων ανατίθεται σε διαφορετικές ομάδες. Αν και συνήθως δεν υπάρχει η δυνατότητα εμπλοκής πολλών διαφορετικών ομάδων ανάπτυξης που δουλεύουν στο ίδιο έργο ούτε είναι εύκολος ο συντονισμός τους, στην πράξη είναι χρήσιμο να αποφεύγεται ο υπερβολικός κατακερματισμός μιας εφαρμογής.

Αν αντιμετωπίζετε με αμφιβολία την τοποθέτηση αυτή, μπορείτε να ξαναδιαβάσετε την Ενότητα 8.1, καθώς και τις Ενότητες 2.1 και 2.2. Αν πάλι τεκμηριωμένα δεν είχατε αμφιβολία, τότε σας αξίζει ένα «μπράβο».

## Άσκηση 3/Κεφάλαιο 8

---

Καταρχάς, ήδη αναφέραμε ότι οι απαιτήσεις I5 και I6 δεν είναι λειτουργικές απαιτήσεις αλλά ορίζουν δύο νέους χειριστές της εφαρμογής.

Ακολουθώντας την οδηγία και λαμβάνοντας υπόψη ότι οι περιπτώσεις χρήσης, όπως ορίστηκαν στην Ενότητα 8.2.2, αντιστοιχούν σε λειτουργικές απαιτήσεις, μπορείτε να οδηγηθείτε στη διάκριση των μη λειτουργικών απαιτήσεων. Αυτές δεν θα αντιστοιχούν σε περιπτώσεις χρήσης.

Εύκολα εντοπίζουμε ότι οι απαιτήσεις αυτές είναι η I και η 6. Η απαίτηση I αφορά το περιβάλλον λειτουργίας ολόκληρης της εφαρμογής και χαρακτηρίζεται ως φυσική απαίτηση.

Η απαίτηση 6 περιγράφει την κατάσταση εισόδου στην περίπτωση χρήσης που αντιστοιχεί στην απαίτηση 5, δηλαδή στην περίπτωση χρήσης με τίτλο «βαθμολόγηση μαθημάτων».

Παρατηρήσεις όπως αυτή δεν είναι πάντα εύκολες και απαιτούν εξοικείωση και εμπειρία, οπότε μην ανησυχείτε αν αυτό το μέρος της απάντησης σας ξέφυγε. Θα μπορούσαμε να έχουμε παρακάμψει τον ύφαλο δίνοντας μια διαφορετική ή μια εκ νέου διατύπωση των απαιτήσεων από το λογισμικό



«Επίκουρος», τότε όμως δεν θα είχαμε τη δυνατότητα αντιπαραβολής ούτε θα επαληθεύαμε τα προβλήματα του πραγματικού κόσμου στον προσδιορισμό των απαιτήσεων από το λογισμικό.

## Άσκηση 4/Κεφάλαιο 8

---

Τα τρία αρχιτεκτονικά στοιχεία του λογισμικού τα οποία εντοπίζονται κατά την ανάλυση είναι:

- Οι κλάσεις που θα αποτελέσουν την εφαρμογή λογισμικού.
- Η ομαδοποίηση αυτών σε πακέτα.
- Οι συσχετίσεις μεταξύ αυτών.

Κατά τη σχεδίαση, η πρώτη εκδοχή της κατασκευαστικής δομής του λογισμικού αναμένεται να μεταβληθεί διότι, προκειμένου να ικανοποιηθούν και οι μη λειτουργικές απαιτήσεις, θα πρέπει να προστεθούν κατασκευαστικές λεπτομέρειες και χαρακτηριστικά του περιβάλλοντος ανάπτυξης και λειτουργίας του λογισμικού.

Συγχαρητήρια σε όσους έδωσαν αμέσως τη σωστή απάντηση. Θα πρέπει να τονιστεί ότι αρχιτεκτονικό στοιχείο του λογισμικού δεν είναι μόνο οι κλάσεις (οι οποίες, σε τελική ανάλυση, συγκροτούν το εκτελέσιμο μέρος της εφαρμογής) και οι σχέσεις τους αλλά και η δόμησή τους σε πακέτα.

## Άσκηση 5/Κεφάλαιο 8

---

Η ζητούμενη κατάταξη έχει ως εξής:

Συνοριακές: 3, 4, 8

Οντοτήτων: 1, 2, 5, 7

Ελέγχου: 6, 9, 10

Από τον ορισμό των κατηγοριών που προηγήθηκε είναι αρκετά φανερό η απάντηση και, ασφαλώς, συγχαρητήρια σε όσους την έδωσαν αμέσως.

## Άσκηση 6/Κεφάλαιο 8

---

Η σχέση υποδηλώνει τη δυνατότητα μέσα σε ένα πακέτο ανάλυσης να υπάρχουν και άλλα τέτοια πακέτα, και το φαινόμενο αυτό να παρατηρείται σε οσαδήποτε επίπεδα βάθους. Σύμφωνα με την περιγραφή της σχέσης συναρμολόγησης, αυτή υποδηλώνει τη δομική ανάλυση μιας σύνθετης οντότητας σε επιμέρους συστατικά. Τα συστατικά αυτά μπορεί να είναι της ίδιας φύσης με τη σύνθετη οντότητα, πετυχαίνοντας έτσι ένα σχήμα ομαδοποίησης πολλών επιπέδων.

Μπράβο σε όσους με την ανάγνωση της απάντησης απλά επιβεβαίωσαν τις υποψίες τους. Οι υπόλοιποι παραπέμπονται στο Κεφάλαιο 7, όπου περιγράφεται η σχέση συναρμολόγησης στην UML.

## Δραστηριότητα 1/Κεφάλαιο 8

---

Προβληματιζόμενοι γύρω από το ερώτημα που τέθηκε, μπορούμε να εκφράσουμε την άποψη ότι ο αναλυτής λογισμικού κατά βάθος έχει στο νου του κάποιους όρους υλοποίησης, όπως περίπου ο αρχιτέκτονας έχει γνώση των δομικών υλικών με τα οποία κατασκευάζεται μια κατοικία. Στη δομημένη ανάλυση οφείλει να συμπεριφερθεί αγνοώντας τους όρους υλοποίησης, πράγμα που, αφενός, δεν είναι εύκολο και, αφετέρου, δυσκολεύει τη διαδικασία απεικόνισης των αποτελεσμάτων της ανάλυσης σε συστατικά στοιχεία υλοποίησης λογισμικού. Το πλεονέκτημα, λοιπόν, της αντικειμενοστρεφούς προσέγγισης είναι ότι ο αναλυτής αποδεσμεύεται από την υποχρέωση να «αγνοεί την υλοποίηση» και το αποτέλεσμα της εργασίας του απεικονίζεται ευκολότερα σε συστατικά στοιχεία υλοποίησης. Αυτό δεν σημαίνει ότι η δουλειά του αναλυτή απλοποιείται ή ότι μπορεί να γίνει πιο αποδοτική, διότι η ανάγκη ευστοχίας στις επιλογές παραμένει, και πάντα η εμπειρία και η δημιουργικότητα θα κρίνουν το αποτέλεσμα της διαδικασίας της ανάλυσης.

## Δραστηριότητα 2/Κεφάλαιο 8

---

Σας προτρέπουμε να χρησιμοποιήσετε εκτενώς το εργαλείο Visual Paradigm για όλες τις ασκήσεις και τις δραστηριότητες του βιβλίου αυτού. Μπορείτε

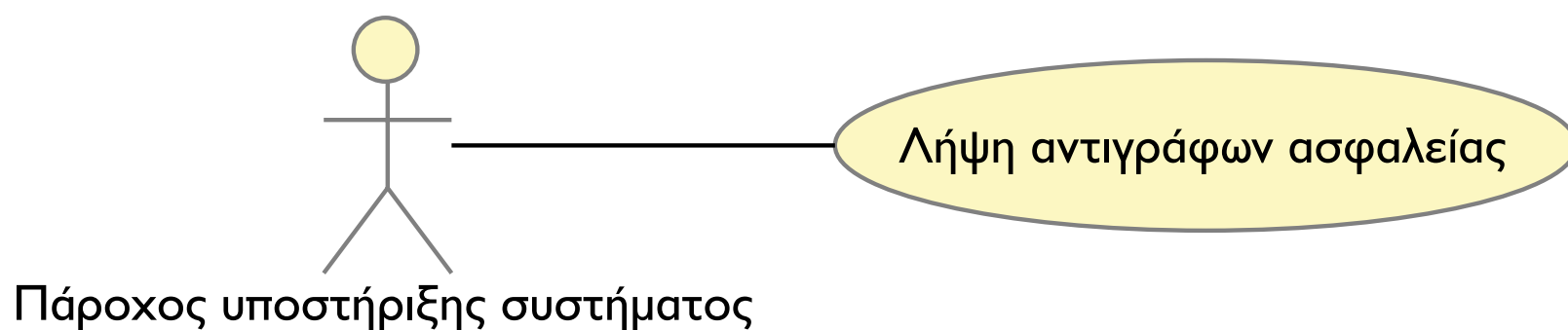
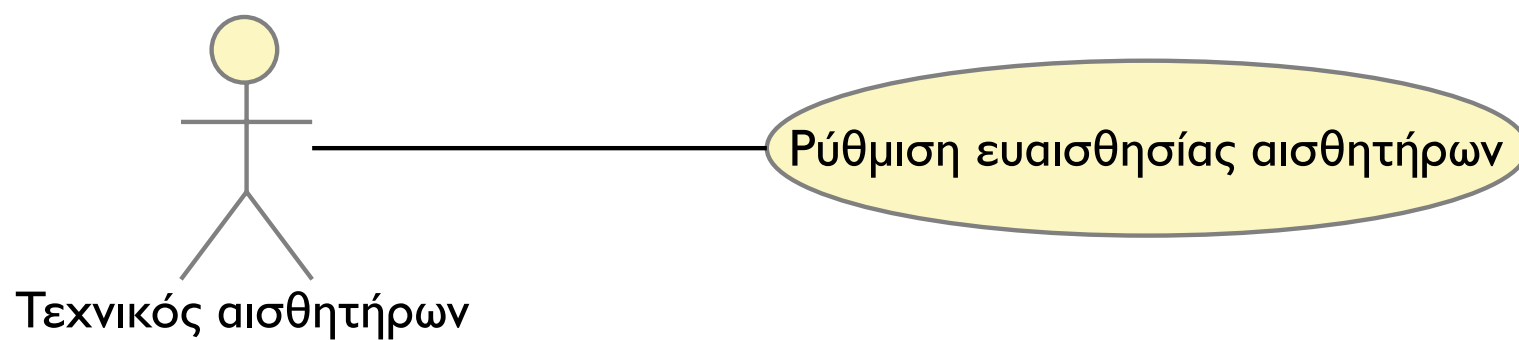
να εγκαταστήσετε τη δωρεάν έκδοση του εργαλείου αυτού από τη σχετική ιστοσελίδα που θα αναζητήσετε στο διαδίκτυο.

### **Δραστηριότητα 3/Κεφάλαιο 8**

---

Είναι φανερό ότι θα προσθέσουμε δύο νέες περιπτώσεις χρήσης που αντιστοιχούν στις δύο νέες λειτουργικές απαιτήσεις. Από την παρατήρηση που ακολουθεί οδηγούμαστε στον εντοπισμό και δύο νέων χειριστών: του «τεχνικού αισθητήρων» και του «παροχέα υποστήριξης συστήματος». Ο πρώτος γνωρίζει πώς να ρυθμίζει την ευαισθησία των αισθητήρων ώστε να δίνουν σωστές μετρήσεις, ενώ ο δεύτερος παρέχει υπηρεσίες υποστήριξης του υπολογιστικού συστήματος. Τα νέα στοιχεία του διαγράμματος περιπτώσεων χρήσης φαίνονται στο Σχήμα 8.26 που ακολουθεί.

**Σχήμα 8.26** Δύο νέες περιπτώσεις χρήσης και δύο νέοι χειριστές για το λογισμικό του παραδείγματος 8.1.



Στο Σχήμα 8.26 είναι χρήσιμο να κάνουμε δύο παρατηρήσεις:

Πρώτο, θα ήταν εύλογο αντί του ορισμού μίας περίπτωσης χρήσης για τη ρύθμιση και των τριών αισθητήρων να είχαμε τρεις, μία για κάθε είδος αισθητήρα. Το θέμα σχετίζεται όχι μόνο με το επίπεδο λεπτομέρειας αλλά και με τη σύλληψη των περιπτώσεων χρήσης. Στις πρώτες φάσεις της ανάπτυξης δεν αποτελεί πρόβλημα ο ορισμός με τον ένα ή τον άλλο τρόπο. Οι απαιτήσεις θα ωριμάσουν καθώς η ανάπτυξη προχωρά, χωρίς αυτό να έχει τις επιπτώσεις που θα είχε στη δομημένη ανάλυση και σχεδίαση.

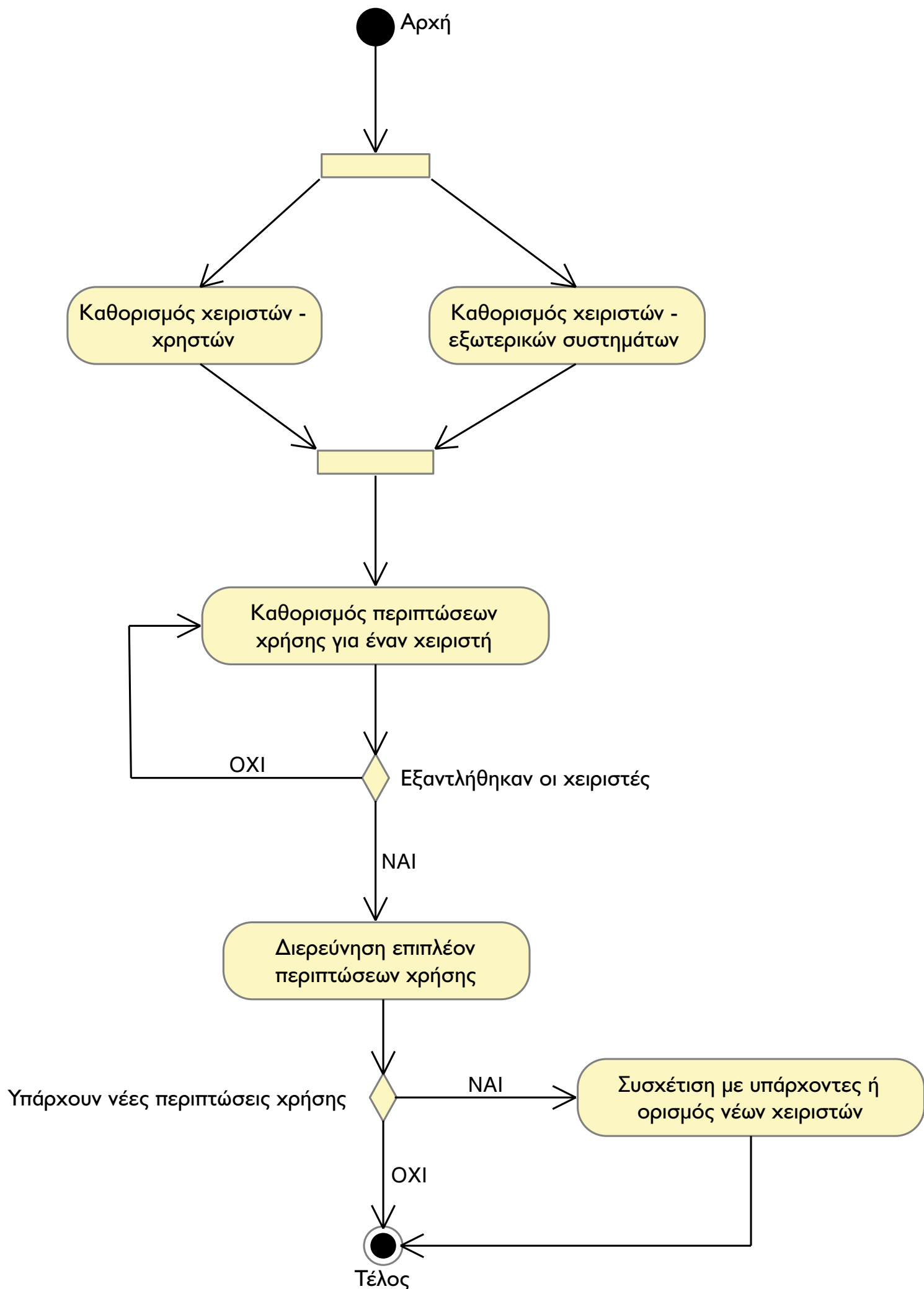
Δεύτερο, δεν είναι σαφές στο σημείο αυτό ποια είναι η διαφορά των δύο νέων χειριστών που ορίσαμε από τον υπάρχοντα χειριστή «μετεωρολόγο». Στο πεδίο του πραγματικού κόσμου, η διαφορά αυτή είναι ασφαλώς σαφής. Ωστόσο, στο πρώιμο στάδιο της ανάπτυξης λογισμικού στο οποίο εν προκειμένω βρισκόμαστε, ίσως δεν διαφαίνεται σε τι διαφέρουν για το λογισμικό οι τρεις χειριστές που αντιστοιχούν σε φυσικά πρόσωπα. Το θέμα θα αντιμετωπιστεί λίγο αργότερα, χωρίς οποιαδήποτε επιλογή μας μέχρι το σημείο αυτό να δημιουργεί πρόβλημα.

Αν μοιραστήκατε μαζί μας τους προβληματισμούς αυτούς πριν διαβάσετε τις δύο τελευταίες παραγράφους, οφείλουμε να σημειώσουμε ότι βρίσκεστε σε πολύ καλό δρόμο. Αν πάλι όχι, μείνετε συντονισμένοι, γιατί τα ενδιαφέροντα ακολουθούν.

## Δραστηριότητα 4/Κεφάλαιο 8

Ένα τέτοιο διάγραμμα φαίνεται στο Σχήμα 8.27 που ακολουθεί. Ο μόνος, ενδεχομένως, πλεονασμός είναι η διάκριση του καθορισμού των χειριστών που αντιστοιχούν σε συστήματα από τον καθορισμό αυτών που αντιστοιχούν σε χρήστες.

**Σχήμα 8.27** Ένα διάγραμμα δραστηριότητας για τον καθορισμό των χειριστών και των περιπτώσεων χρήσης.



Μπορείτε να δώσετε και άλλα τέτοια διαγράμματα που να αποδίδουν τη δική σας ερμηνεία για το ζητούμενο.

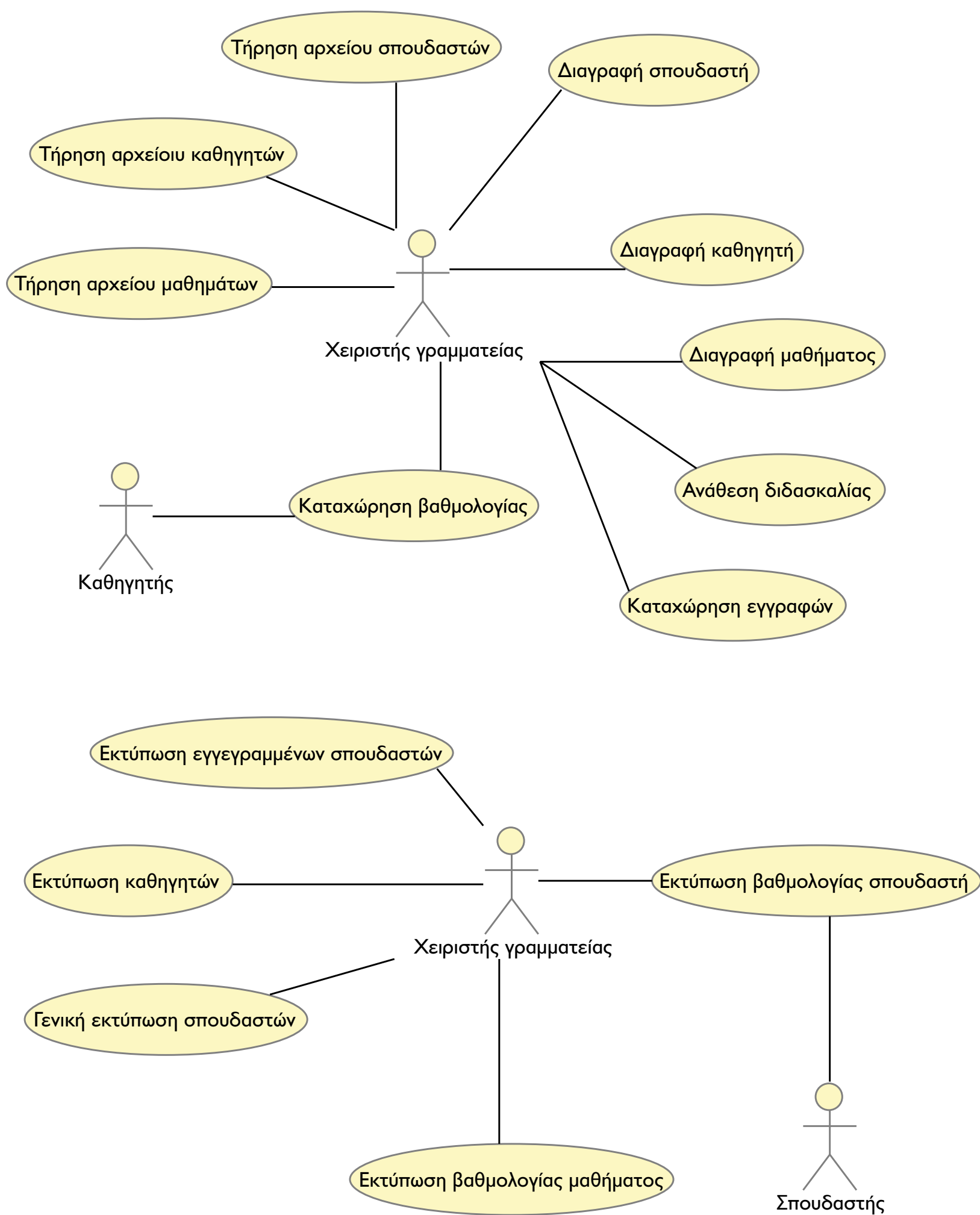
## Δραστηριότητα 5/Κεφάλαιο 8

---

Διαπιστώνουμε ότι δεν πρόκειται για νέες λειτουργικές απαιτήσεις που αντιστοιχούν σε νέες περιπτώσεις χρήσης αλλά για τον εντοπισμό δύο νέων χειριστών του συστήματος οι οποίοι αντιστοιχούν σε χρήστες – φυσικά πρόσωπα.

Στο Σχήμα 8.28 που ακολουθεί φαίνεται η δική μας εκδοχή του πλήρους –μέχρι το σημείο αυτό– διαγράμματος περιπτώσεων χρήσης για όλες τις λειτουργικές απαιτήσεις της εφαρμογής «Επίκουρος». Κάποιοι χειριστές εμφανίζονται περισσότερες από μία φορές για λόγους αναγνωσιμότητας του διαγράμματος.

**Σχήμα 8.28** Το μοντέλο περιπτώσεων χρήσης για το λογισμικό «Επίκουρος».





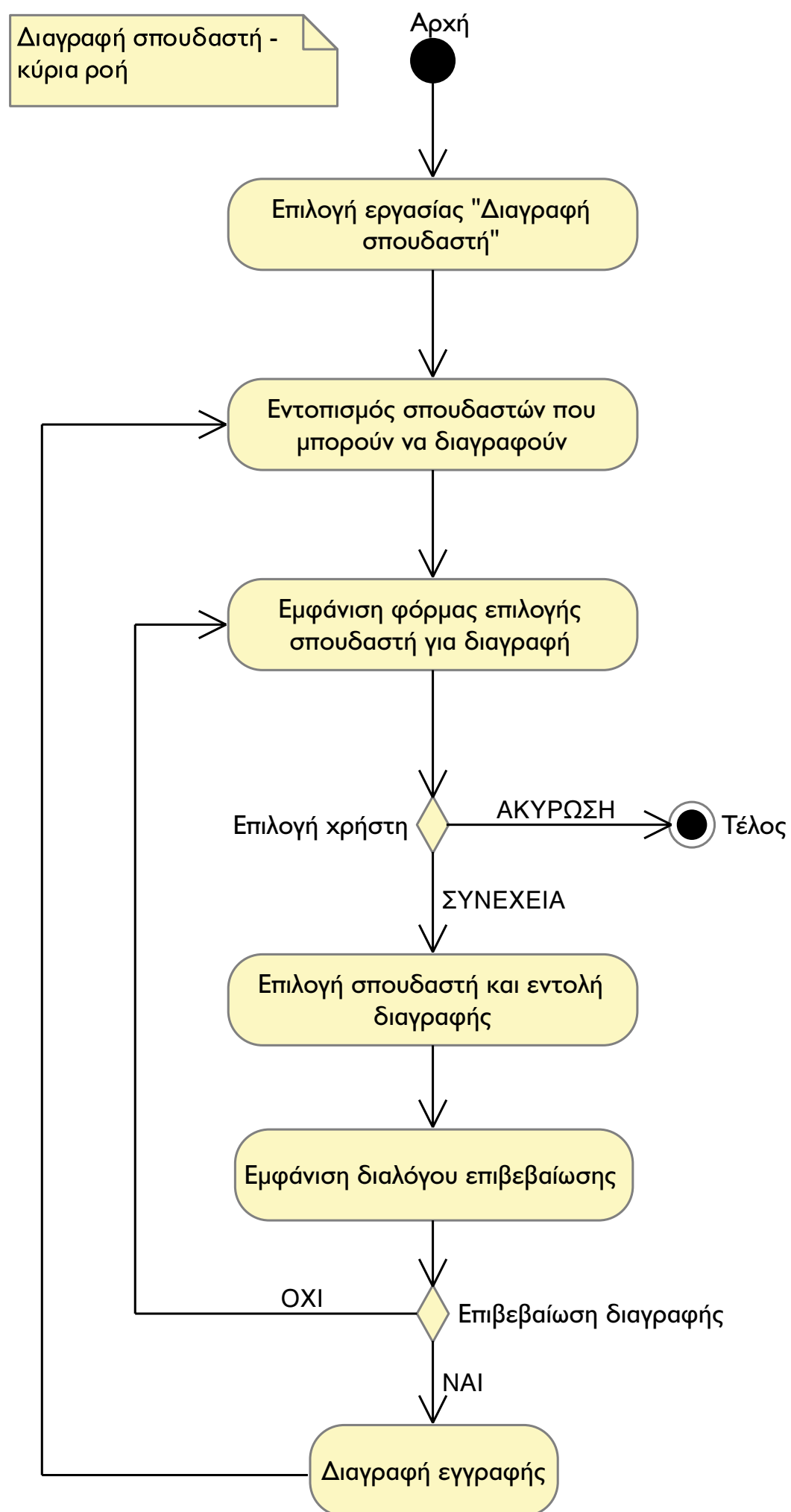
Όπως μπορείτε να αντιληφθείτε, το διάγραμμα αυτό δεν σχεδιάστηκε με κάποιο σχεδιαστικό πακέτο, αλλά με τη χρήση του εργαλείου Visual Paradigm. Ενθαρρύνεστε να κάνετε και εσείς το ίδιο.

## **Δραστηριότητα 6/Κεφάλαιο 8**

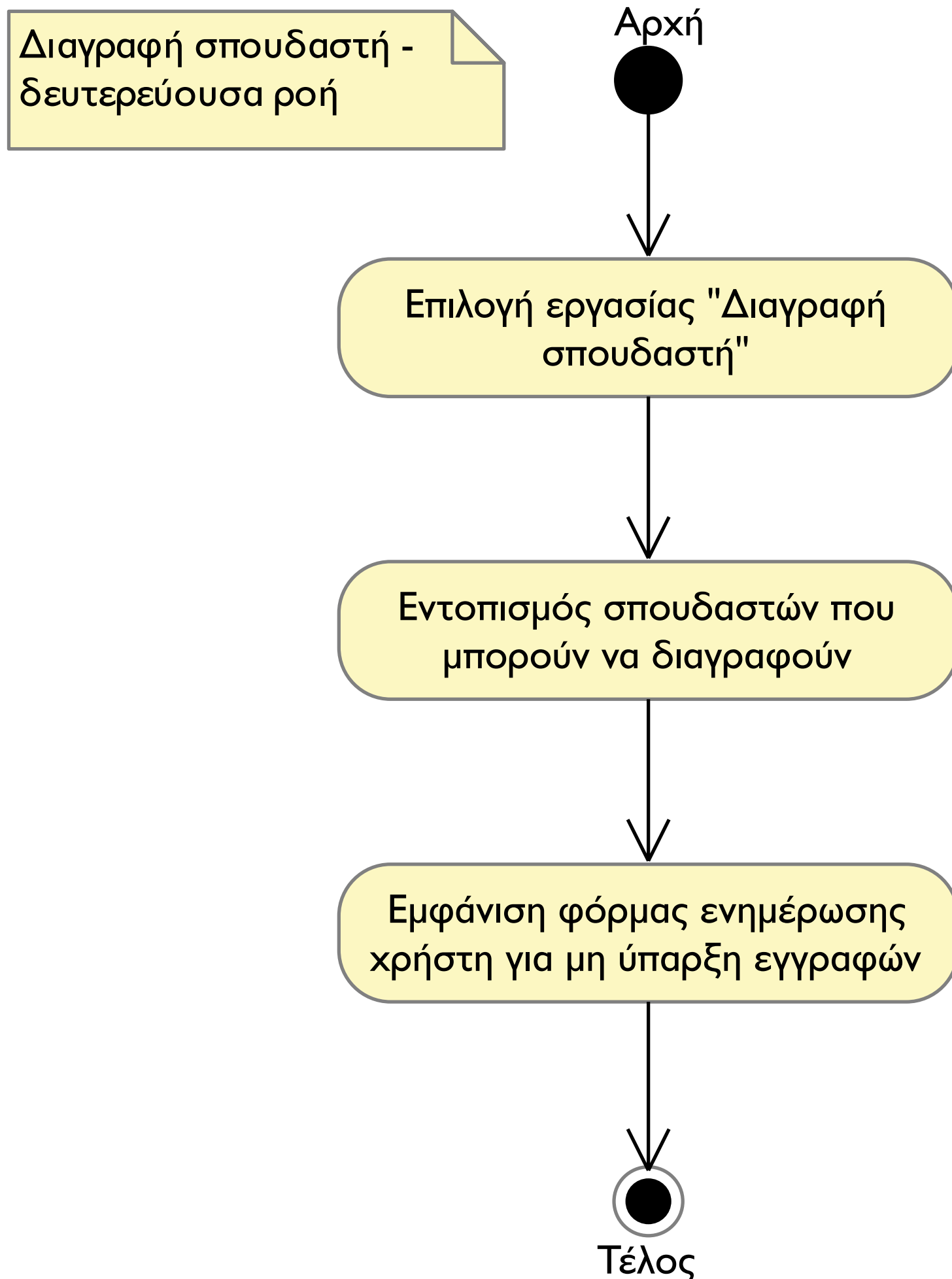
---

Θα διακρίνουμε στη δική μας απάντηση την κύρια από τη δευτερεύουσα ροή. Το ζητούμενο διάγραμμα για την κύρια ροή δίνεται στο Σχήμα 8.29, ενώ για τη δευτερεύουσα στο Σχήμα 8.30.

**Σχήμα 8.29** Ένα διάγραμμα δραστηριότητας για την κύρια ροή της περίπτωσης χρήσης «διαγραφή σπουδαστή» του λογισμικού «Επίκουρος».



**Σχήμα 8.30** Ένα διάγραμμα δραστηριότητας για τη δευτερεύουσα ροή της περίπτωσης χρήσης «διαγραφή σπουδαστή» του λογισμικού «Επίκουρος».



Μπορείτε ασφαλώς να σκεφτείτε ότι δεν απαιτείται ξεχωριστό διάγραμμα για τη δευτερεύουσα ροή και ότι το θέμα μπορούσε να τακτοποιηθεί με έναν ακόμη κόμβο απόφασης στο διάγραμμα που αφορά την κύρια ροή. Ασφαλώς! Στη μελέτη περίπτωσης που μας απασχολεί εδώ τα πράγματα είναι σχετικά απλά. Να μην ξεχνάτε όμως ότι στον πραγματικό κόσμο τα πράγματα μπορεί να είναι πολύ πιο δύσκολα, οπότε η χρήση πολλών διαγραμμάτων δεν είναι απίθανη. Από την άλλη, αν τα πράγματα γίνουν πολύ περίπλοκα, ίσως να πρέπει να ξανασκεφτεί κανείς τον ορισμό της περίπτωσης χρήσης και τη διάσπασή της σε περισσότερες από μία.

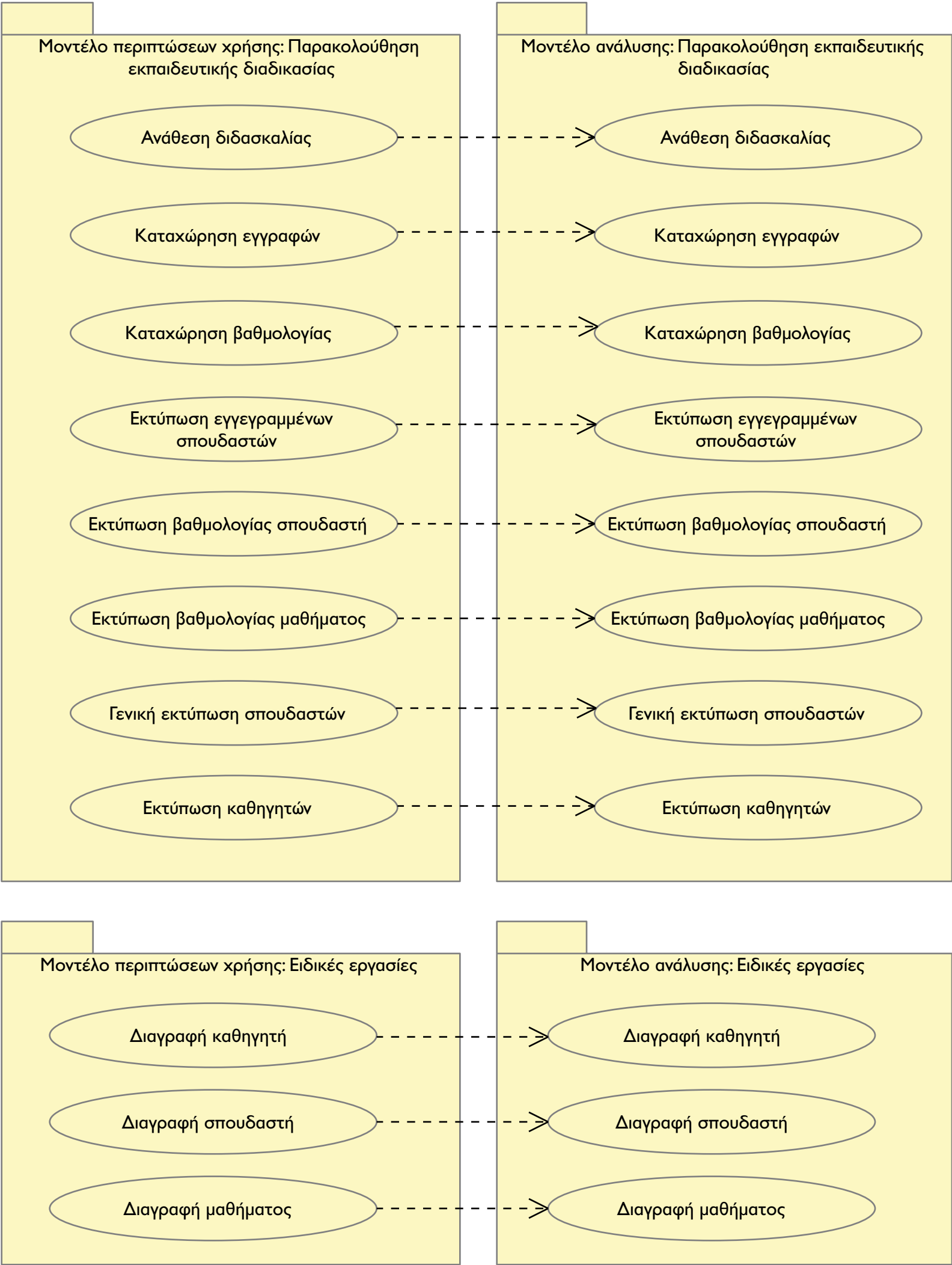
Μην ανησυχείτε αν αυτά σας φαίνονται βουνό. Η εμπειρία και η προσαρμογή σε έναν συγκεκριμένο τρόπο δουλειάς δίνουν πάντα τη λύση.

## Δραστηριότητα 7/Κεφάλαιο 8

---

Πρόκειται για μάλλον πολύ εύκολη εργασία, την οποία ενθαρρύνεστε έντονα να πραγματοποιήσετε χρησιμοποιώντας το εργαλείο Visual Paradigm. Τα ζητούμενα διαγράμματα φαίνονται στο Σχήμα 8.31.

**Σχήμα 8.3Ι** Από τις περιπτώσεις χρήσης στις περιπτώσεις χρήσης πεδίου ανάλυσης για την εφαρμογή λογισμικού «Επίκουρος».

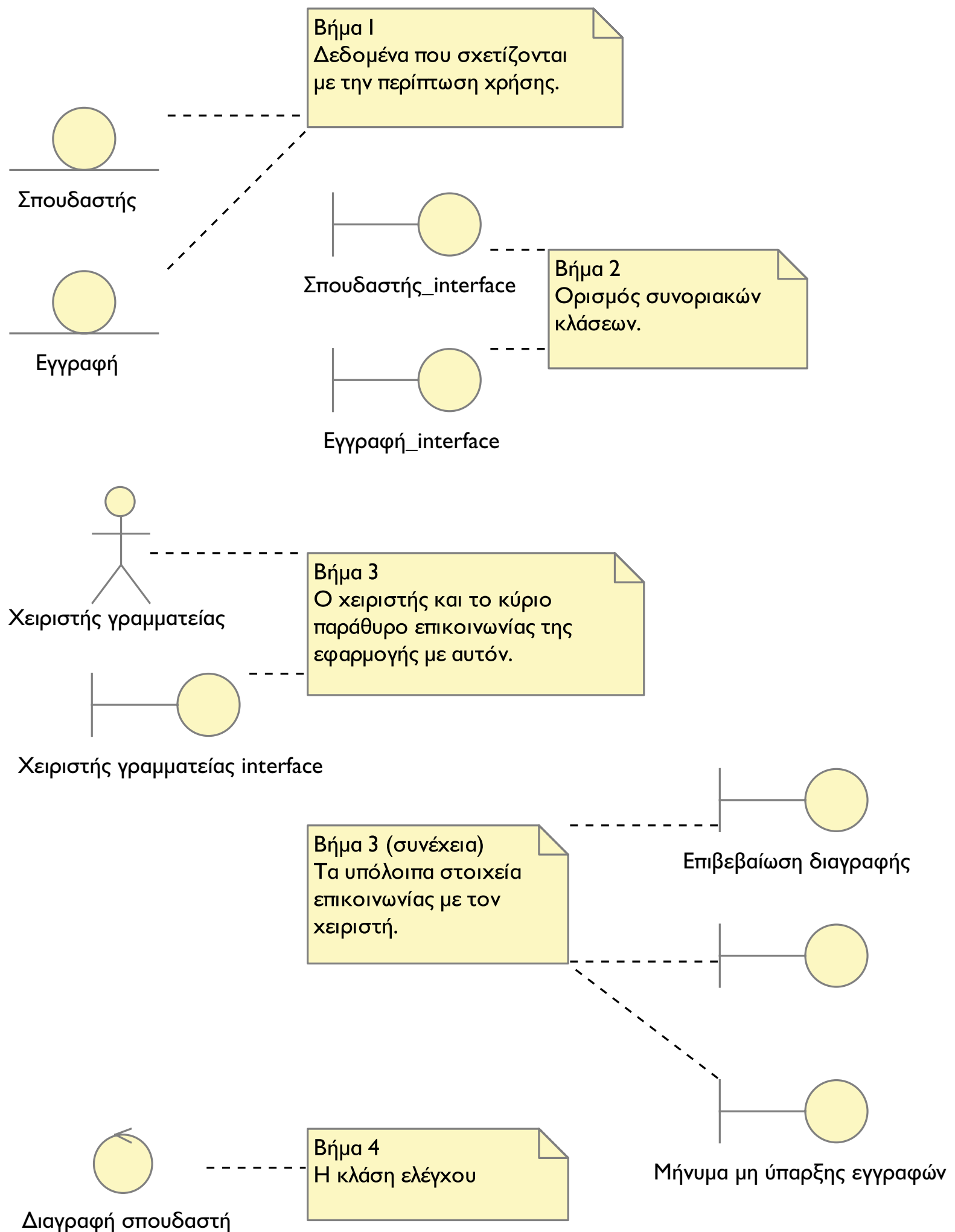


## Δραστηριότητα 8/Κεφάλαιο 8

---

Λαμβανομένων υπόψη των υποδείξεων, εμείς προτείνουμε τις κλάσεις που φαίνονται στο Σχήμα 8.32. Παρατηρήστε ότι μέχρι το σημείο αυτό δεν κάνουμε τίποτε άλλο από το να εφαρμόζουμε με ακρίβεια τα βήματα που περιγράφηκαν. Είναι φυσικό να έχετε απορίες όπως, για παράδειγμα, τι ρόλο παίζουν οι συνοριακές κλάσεις που ορίσαμε για κάθε κλάση οντοτήτων, και μάλιστα να κατευθύνεστε σε απαντήσεις που σχετίζονται με την κατασκευή του προγράμματος. Ή απορίες του τύπου «ταυτίζεται η κλάση τάδε, την οποία εντόπισα κατά την ανάλυση και των δύο περιπτώσεων χρήσης ή πρόκειται για διαφορετικές κλάσεις;». Η απάντηση είναι ότι κανένα τέτοιο ερώτημα δεν αφορά την ανάλυση αλλά τη σχεδίαση, οπότε ας αφήσουμε τα πράγματα να εξελιχθούν.

**Σχήμα 8.32** Ανάλυση της περίπτωσης χρήσης «διαγραφή σπουδαστή» της εφαρμογής «Επίκουρος» (I).



Συγχαρητήρια σε όσους εντόπισαν τις ίδιες κλάσεις (ασφαλώς, δεν περιμένουμε να έχετε σκεφτεί και τα ίδια ονόματα). Περισσότερα συγχαρητήρια όμως σε όσους αντιμετωπίζουν τέτοια προβλήματα χρησιμοποιώντας το εργαλείο Visual Paradigm, ακόμη και αν τους ξεφεύγουν πράγματα. Μια επανάληψη με προσεκτική μελέτη των παραδειγμάτων πάντα έχει κάτι να προσθέσει.

## Δραστηριότητα 9/Κεφάλαιο 8

---

Ο σχολιασμός είναι ανάλογος με αυτόν που δώσαμε στη μελέτη περίπτωσης. Ο χρήστης επιλέγει την εργασία (I) και αρχικοποιείται η μονάδα ελέγχου της περίπτωσης χρήσης (2).

Στη συνέχεια, προκειμένου να διαπιστωθεί αν υπάρχουν σπουδαστές των οποίων η διαγραφή επιτρέπεται, τρέχει μια ερώτηση προς το αρχείο εγγραφών σπουδαστών από την οποία προκύπτει η λίστα των σπουδαστών (Σ1) που έχουν εγγραφεί σε μαθήματα (3, 4). Οι σπουδαστές που δεν έχουν εγγραφεί είναι εκείνοι που αναφέρονται στο αρχείο σπουδαστών, όχι όμως στο αρχείο εγγραφών. Οπότε εκτελείται αντίστοιχη ερώτηση προς το αρχείο σπουδαστών (5, 6) και προκύπτει η λίστα Σ2.

Σύμφωνα με την περιγραφή της ροής, οι Σ1 και Σ2 ταυτίζονται, δηλαδή όλοι οι σπουδαστές έχουν εγγραφεί σε μαθήματα, οπότε δεν επιτρέπεται η διαγραφή κανενός. Οπότε, το μόνο που μένει είναι να ενημερωθεί σχετικά ο χρήστης με την εμφάνιση διαλόγου.

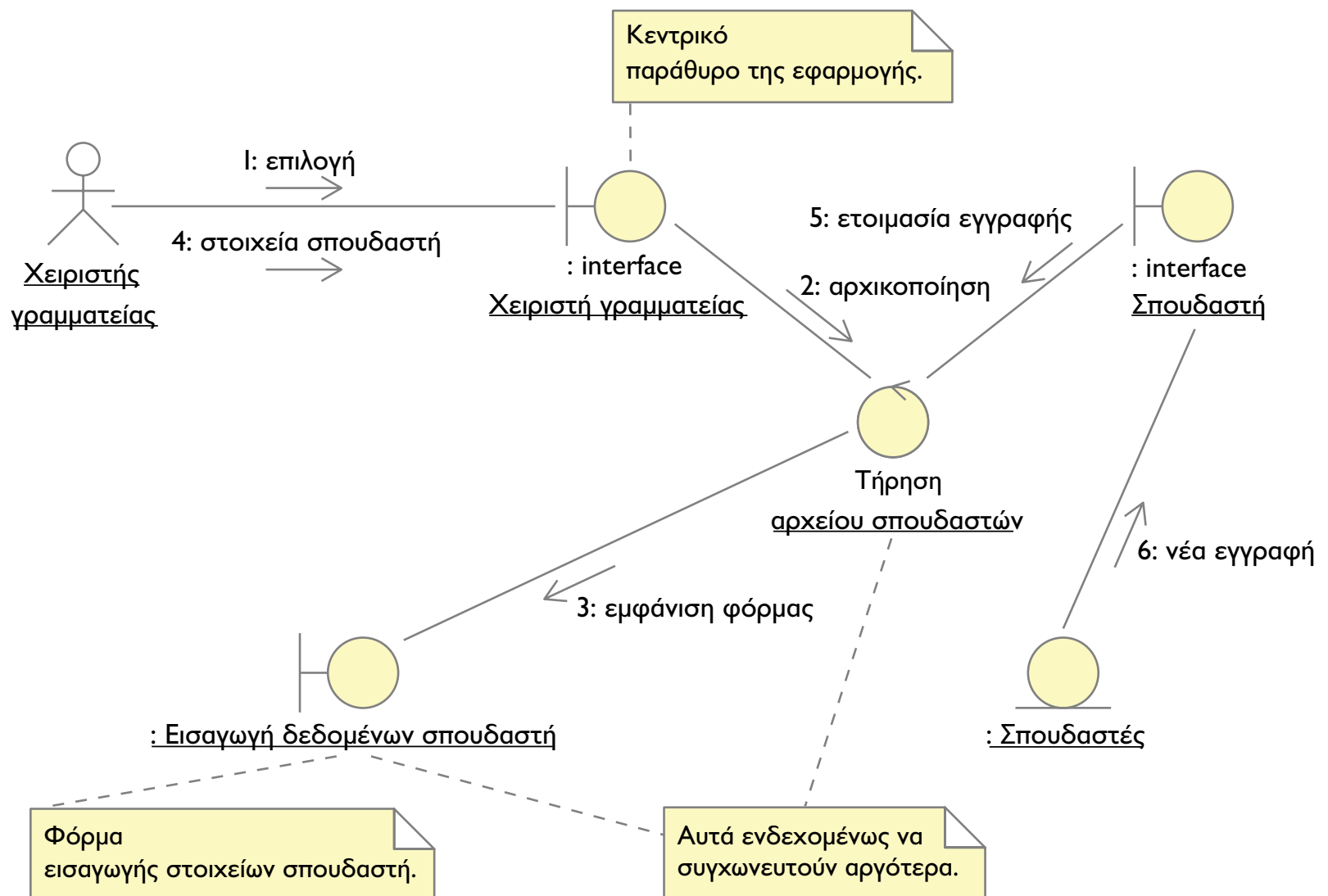
## Δραστηριότητα 10/Κεφάλαιο 8

---

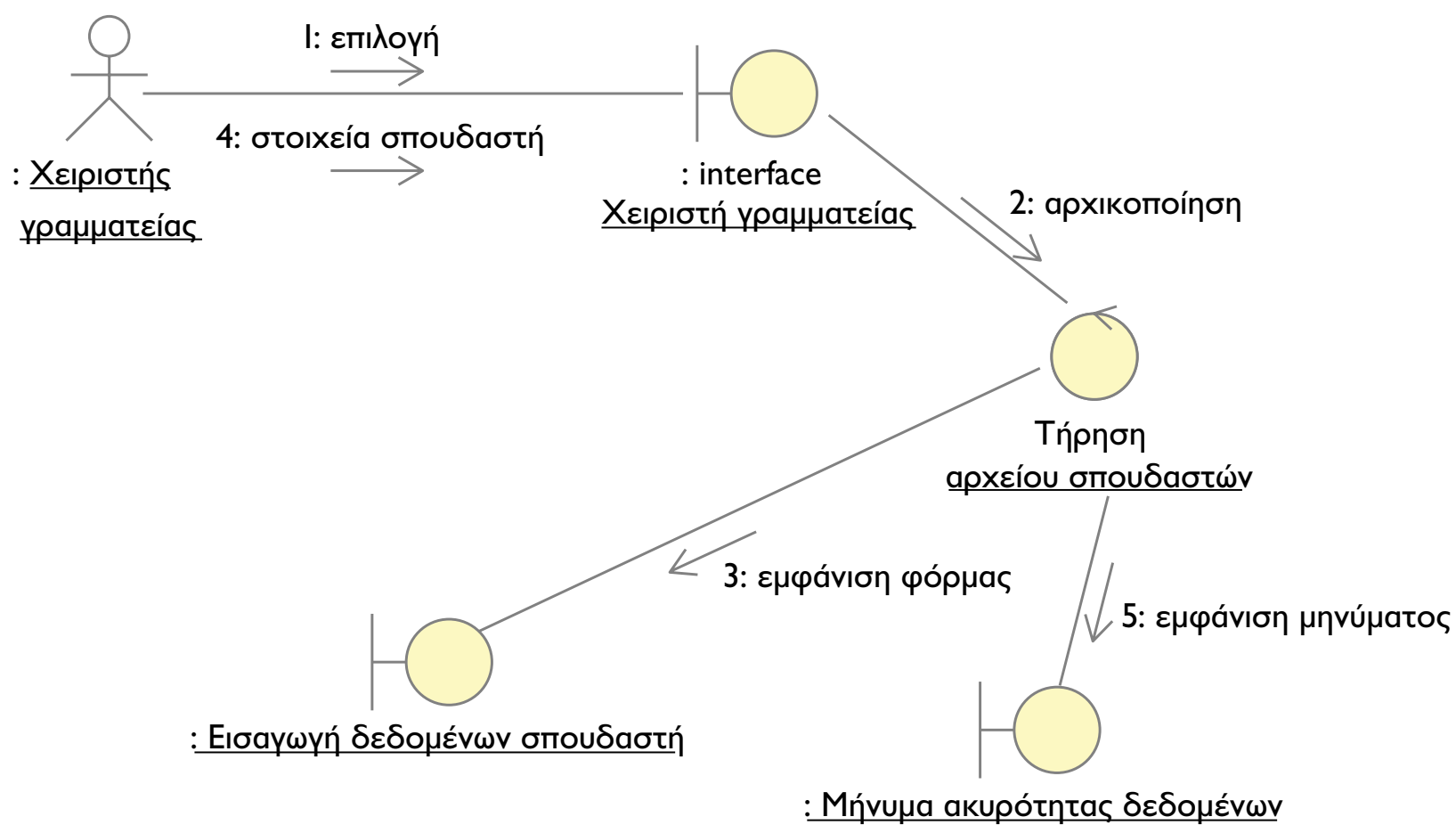
Τα ζητούμενα διαγράμματα, σύμφωνα με τη δική μας πάντα εκδοχή, δίνονται στη συνέχεια. Στο Σχήμα 8.33 φαίνεται εκείνο που αντιστοιχεί στην κύρια ροή, ενώ στο Σχήμα 8.34 εκείνο που αντιστοιχεί στη δευτερεύουσα.



**Σχήμα 8.33** Διάγραμμα συνεργασίας για την κύρια ροή της περίπτωσης χρήσης «τήρηση αρχείου σπουδαστών».



**Σχήμα 8.34** Διάγραμμα συνεργασίας για τη δευτερεύουσα ροή της περίπτωσης χρήσης «τήρηση αρχείου σπουδαστών».



## ΒΙΒΛΙΟΓΡΑΦΙΑ

Booch G., Rumbaugh J., Jacobson I., *The Unified Modeling Language User Guide*, Addison-Wesley.

Booch, G., *Object-Oriented Analysis and Design with Applications*, Addison-Wesley.

Fowler M., Scott K., *UML Distilled*, Addison-Wesley.

Henderson-Sellers B., *A Book of Object-Oriented Knowledge*, Prentice Hall.

Jacobson I., Booch G., Rumbaugh J., *The Unified Software Development Process*, Addison-Wesley.

Jacobson I., Christerson M., Johnson P., Overgaard G., *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley.

Martin J., Odell J., *Object-Oriented Analysis and Design*, Prentice Hall.

Pressman, R. S., *Software Engineering-A Practitioners Approach*, McGraw-Hill.

Quadrani T., *Visual Modeling with Rational Rose and UML*, Addison-Wesley.

Rumbaugh J., Jacobson I., Booch G., *The Unified Modeling Language Reference Manual*, Addison-Wesley.

Rumbaugh J., *Object-Oriented Modeling and Design*, Prentice Hall.

Schneider G., Winters J., *Applying Use Cases: A practical Guide*, Addison-Wesley.

Sommerville, I. *Software Engineering*, London: Addison-Wesley.

*Use Case Driven Object Modeling with UML: A Practical Approach*, Doug Rosenberg, Addison-Wesley, ISBN 978-0201432893

## ΕΙΣΑΓΩΓΗ ΣΤΗΝ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗ ΣΧΕΔΙΑΣΗ

Σκοπός του κεφαλαίου είναι να εισάγει τον αναγνώστη στην αντικειμενοστρεφή σχεδίαση, συνεχίζοντας την αναφορά στην ενοποιημένη προσέγγιση ανάπτυξης λογισμικού.

Μετά τη μελέτη του κεφαλαίου, ο αναγνώστης θα είναι σε θέση:

- να κατασκευάζει UML διαγράμματα κλάσεων σχεδίασης ξεκινώντας από τις κλάσεις του πεδίου ανάλυσης,
- να κατασκευάζει UML διαγράμματα αλληλουχίας ή ακολουθίας ή σειράς (sequence diagrams) ξεκινώντας από τις αντίστοιχες περιγραφές συνεργασίας του πεδίου ανάλυσης,
- να ορίζει υποσυστήματα και να αντιστοιχεί σε αυτά τα συστατικά στοιχεία λογισμικού (components) που θα αποτελέσουν την υπό ανάπτυξη εφαρμογή,
- να κατασκευάζει το μοντέλο διάταξης ή εγκατάστασης (deployment model) μιας εφαρμογής λογισμικού,
- να ακολουθεί συγκεκριμένα βήματα ώστε να μεταβαίνει από το μοντέλο ανάλυσης στο μοντέλο σχεδίασης μιας εφαρμογής λογισμικού σύμφωνα με την ενοποιημένη προσέγγιση ανάπτυξης λογισμικού.

### Έννοιες-κλειδιά

---

- *Κλάση σχεδίασης*
- *Διάγραμμα αλληλουχίας ή ακολουθίας ή σειράς UML (sequence diagram)*
- *Σύστημα – υποσύστημα*
- *Μοντέλο διάταξης ή εγκατάστασης UML (deployment model)*

- Συστατικό στοιχείο (component)
- Μοντέλο σχεδίασης

## Σύνοψη

---

Η αντικειμενοστρεφής σχεδίαση είναι μια δημιουργική διαδικασία μετάβασης από το πεδίο του προβλήματος στο πεδίο της λύσης του. Σε αντιδιαστολή με τη δομημένη σχεδίαση, χρησιμοποιεί ως επί το πλείστον τα ίδια δομικά συστατικά λογισμικού: τις κλάσεις που έχουν ήδη «αποκαλυφθεί» κατά την ανάλυση, τις οποίες όμως εξειδικεύει σε επίπεδο κατασκευαστικής λεπτομέρειας. Τόσο η λεπτομερής πειθαρχία των διαδικασιών που ακολουθούνται όσο και η περιγραφή των αποτελεσμάτων τους είναι αντικείμενο που σχετίζεται με το εκάστοτε περιβάλλον ανάπτυξης λογισμικού.

Η UML μάς προσφέρει ένα σύνολο από εκφραστικά εργαλεία για την παράσταση των αποτελεσμάτων της σχεδίασης. Εκτός από τα ήδη γνωστά μας διαγράμματα κλάσεων και συνεργασίας, εισάγεται το διάγραμμα αλληλουχίας ή ακολουθίας ή σειράς, το διάγραμμα διάταξης και το διάγραμμα συστατικών. Το επίπεδο λεπτομέρειας κατά τη δημιουργία των διαγραμμάτων αυτών εξαρτάται από το περιβάλλον και τον τρόπο εργασίας κάθε σχεδιαστή. Σε κάθε περίπτωση, ένα διάγραμμα κλάσεων με «μικρή λεπτομέρεια» παραμένει διάγραμμα κλάσεων το οποίο, μολονότι δεν είναι κυριολεκτικά υλοποιήσιμο, εξακολουθεί να καθοδηγεί τον προγραμματιστή στην εργασία του. Όσο πιο πλήρες είναι το μοντέλο σχεδίασης και περιέχει, λόγου χάρη, συστατικά στοιχεία που κάνουν απολύτως αντιληπτή τη συμπεριφορά του λογισμικού (π.χ. διαγράμματα ακολουθίας), τόσο λεπτομερέστερη είναι η σχεδίαση. Σε κάθε περίπτωση, η προσέγγιση που ακολουθούμε εδώ οριοθετείται από τον εκπαιδευτικό της χαρακτήρα και δεν στοχεύει στην πλήρη κάλυψη όλων των πλευρών του θέματος, για την οποία ο αναγνώστης παραπέμπεται στη βιβλιογραφία.

## Εισαγωγικές παρατηρήσεις

---

Συνεχίζοντας την αναφορά στην αντικειμενοστρεφή ανάπτυξη λογισμικού, φτάνουμε στην αντικειμενοστρεφή σχεδίαση σύμφωνα με την ενοποιημένη προσέγγιση ανάπτυξης λογισμικού στην οποία αναφερόμαστε στο βιβλίο αυτό. Η σχεδίαση

λογισμικού κατά την αντικειμενοστρεφή φιλοσοφία οδηγεί στον καθορισμό κλάσεων οι οποίες θα υλοποιηθούν προγραμματιστικά, αντικείμενα των οποίων θα εκδηλώνουν την επιθυμητή συμπεριφορά κατά την εκτέλεση της εφαρμογής λογισμικού. Λέγοντας «καθορισμός» εννοούμε την περιγραφή των πεδίων και των μεθόδων των κλάσεων, καθώς και όλες τις απαραίτητες κατασκευαστικές λεπτομέρειες για τη συγγραφή του πηγαίου κώδικα που αντιστοιχεί σε αυτές.

Για όσους έχουν ασχοληθεί με την ανάπτυξη λογισμικού με τη δομημένη ανάλυση και σχεδίαση, υπάρχει μια σύγχυση: ενώ στη δομημένη ανάλυση και σχεδίαση στη φάση της ανάλυσης κάνουμε λόγο για άλλες έννοιες απ' ό,τι στη φάση της σχεδίασης, στην αντικειμενοστρεφή προσέγγιση δεν ισχύει το ίδιο. Κατά τη δομημένη ανάλυση μιλάμε για προδιαγραφές, διάγραμμα ροής δεδομένων, λεξικό δεδομένων και, ενδεχομένως, για διάγραμμα καταστάσεων (Κεφάλαια 4 και 5). Ακολουθώντας, κατά τη δομημένη σχεδίαση μιλάμε για διάγραμμα δομής προγράμματος και για ψευδοκώδικα. Δεν ισχύει το ίδιο για την αντικειμενοστρεφή ανάλυση και σχεδίαση. Στην περίπτωση αυτή, τόσο κατά την ανάλυση όσο και κατά τη σχεδίαση μιλάμε για κλάσεις: «κλάσεις ανάλυσης» και «κλάσεις σχεδίασης», όπως τις ονομάζουμε.

Ωστόσο, δεν είναι πάντα σαφές πότε μια κλάση παύει να είναι κλάση ανάλυσης και γίνεται κλάση σχεδίασης. Μια λεπτομερής αντικειμενοστρεφής ανάλυση οδηγεί σε ένα μοντέλο ανάλυσης που περιέχει κλάσεις οι οποίες θα μπορούσαν κάλλιστα να αποτελούν κλάσεις σχεδίασης μιας λιγότερο λεπτομερούς αντικειμενοστρεφούς σχεδίασης. Αν το καλοσκεφτούμε, η σύγχυση αυτή έχει μικρό νόημα και δεν θα πρέπει να δεσπόζει στην πρακτική ενός κατασκευαστή λογισμικού. Όπως κατά τη δομημένη ανάλυση και σχεδίαση που στην πράξη φτάνουμε μέχρι ενός επιπέδου λεπτομέρειας από το οποίο και μετά «αφήνουμε τη δουλειά στον προγραμματιστή», έτσι και στην περίπτωση της αντικειμενοστρεφούς τεχνολογίας. Το ποιο είναι το επίπεδο αυτό εξαρτάται από πολλούς παράγοντες, μεταξύ των οποίων το κόστος και η «προγραμματιστική κουλτούρα», αλλά και τα πρότυπα που επιβάλλεται κατά περίπτωση να ακολουθήσουμε. Τα πράγματα είναι πιο αυστηρά για εφαρμογές «κρίσιμης αποστολής» (*mission critical applications*) και λιγότερο για άλλες περιπτώσεις.

Η διάκριση μιας κλάσης σε κλάση ανάλυσης ή σχεδίασης αποτελεί πηγή σύγχυσης και για τους έχοντες εμπειρία στην αντικειμενοστρεφή ανάπτυξη. Ωστόσο, οι τελευταίοι έχουν πιο ισχυρή συνείδηση του ότι ακριβώς αυτή η εκφραστική δύναμη



των αντικειμένων ως εργαλείο μοντελοποίησης του κόσμου, όπως θα τονίσουμε επανειλημμένως στη συνέχεια, είναι που δημιουργεί την όποια σύγχυση η οποία, επαναλαμβάνουμε, έχει μικρό νόημα.

## ΕΝΟΤΗΤΑ 9.1. Η ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗΣ ΣΧΕΔΙΑΣΗ ΣΤΟ ΜΟΝΤΕΛΟ ΚΥΚΛΟΥ ΖΩΗΣ

Σύμφωνα με το μοντέλο κύκλου ζωής της ενοποιημένης προσέγγισης ανάπτυξης λογισμικού, η αντικειμενοστρεφής σχεδίαση ακολουθεί με τρόπο «φυσικό» την ανάλυση σε μια σειρά διαδικασιών που μας οδηγούν από το αρχικό πρόβλημα στην επίλυσή του. Παρότι η εξέλιξη αυτή είναι αναμενόμενη και κατ' αντιστοιχία με τις διαδικασίες της δομημένης ανάλυσης και σχεδίασης, στην αντικειμενοστρεφή της εκδοχή κρύβει μια σημαντική διαφορά, την οποία τονίσαμε στις εισαγωγικές παρατηρήσεις: η αντικειμενοστρεφής ανάλυση και η αντικειμενοστρεφής σχεδίαση είναι διαδικασίες τόσο κοντινές που πολλοί μηχανικοί λογισμικού τις αντιμετωπίζουν ως μια ενιαία διαδικασία στο πλαίσιο πάντα του εκάστοτε αντικειμένου, της κουλτούρας της ομάδας ανάπτυξης και των λοιπών περιορισμών ή απαιτήσεων της ανάπτυξης. Η εγγύτητα αυτή των δύο διαδικασιών είναι συνέπεια των αυξημένων, σε σχέση με τη δομημένη προσέγγιση, δυνατοτήτων μοντελοποίησης του φυσικού κόσμου από κλάσεις και αντικείμενα. Προκύπτει, δηλαδή, από την κεντρική αξία της αντικειμενοστρεφούς φιλοσοφίας. Σε αυτό το σημείο θα ήταν χρήσιμο να παραθέσουμε τους ορισμούς των διαδικασιών της αντικειμενοστρεφούς ανάλυσης και σχεδίασης, έτσι ώστε ο λόγος της εγγύτητας και της αλληλεξάρτησής τους να γίνει πιο εμφανής:

### Αντικειμενοστρεφής ανάλυση:

Η αντικειμενοστρεφής ανάλυση ασχολείται με την ανάπτυξη ενός αντικειμενοστρεφούς μοντέλου του **πεδίου προβλήματος**. Τα αντικείμενα που προκύπτουν από την ανάλυση αντιστοιχούν σε οντότητες και μεθόδους συνυφασμένες με το αρχικό πρόβλημα προς επίλυση.

### Αντικειμενοστρεφής σχεδίαση:

Η αντικειμενοστρεφής σχεδίαση ασχολείται με την ανάπτυξη ενός αντικειμενοστρεφούς μοντέλου **ενός συστήματος λογισμικού** το οποίο θα υλοποιήσει τις προκαθορισμένες λειτουργικές απαιτήσεις του προβλήματος.

Ας σημειωθεί ότι δεν πρόκειται για διαφορά διατύπωσης αλλά ουσίας: Κατά τη διαδικασία της αντικειμενοστρεφούς σχεδίασης μπορεί να ανακαλύψουμε πως κάποια αντικείμενα του πεδίου προβλήματος (δηλαδή που προέκυψαν από την ανάλυση) είναι πολύ «συναφή» ή και όμοια με αντίστοιχα αντικείμενα του σχεδίου της επίλυσης (δηλαδή που προκύπτουν κατά τη σχεδίαση). Στον αντίποδα, κάποια άλλα (στην καλύτερη περίπτωση λίγα) αντικείμενα που εντοπίστηκαν από την ανάλυση μπορεί να διαφέρουν με αυτά που επιτελούν τις λειτουργίες τους στο σχεδιαστικό μοντέλο. Σε κάθε περίπτωση πάντως, είναι αναμενόμενο ο σχεδιαστής να χρειαστεί να προσθέσει καινούρια αντικείμενα ή/και να αναπτύξει περαιτέρω τα υπάρχοντα στο σχεδιαστικό μοντέλο. Η πρόσθεση καινούριων αντικειμένων και η προσαρμογή των υπάρχοντων στο σχεδιαστικό μοντέλο γίνεται βάσει των χαρακτηριστικών του εγχειρήματος και της κατανόησης του προβλήματος από τον ίδιο τον σχεδιαστή. Υλικό απαραίτητο για την κατανόηση αυτή έχει κατασκευαστεί ήδη από τη φάση της ανάλυσης. Αφού, λοιπόν, η αρχική κατανόηση και μοντελοποίηση του προβλήματος, όπως και η μετέπειτα σχεδίαση του λογισμικού προς υλοποίηση, γίνεται σε επίπεδο αντικειμένων,



είναι φυσικό ο σχεδιαστής πολλές φορές να χρειάζεται να μεταπηδά από το στάδιο της σχεδίασης στο στάδιο της ανάλυσης και το αντίστροφο.

Από τα παραπάνω συμπεραίνεται η μεγάλη σημασιολογική εγγύτητα της ανάλυσης με τη σχεδίαση στα πλαίσια της αντικειμενοστρεφούς φιλοσοφίας. Αυτή η εγγύτητα προκύπτει κυρίως από τη δυνατότητα που μας παρέχει η ενοποιημένη προσέγγιση ανάπτυξης λογισμικού στο να μεταφέρουμε προβλήματα του φυσικού κόσμου σε λεπτομερή αντικειμενοστρεφή μοντέλα μέσω του εντοπισμού έγκυρων περιπτώσεων χρήσης. Στην περίπτωση της ενοποιημένης προσέγγισης ανάπτυξης λογισμικού, η μετάβαση από την ανάλυση στη σχεδίαση επιτυγχάνεται με τη χρήση κοινών εργαλείων της UML και απαιτεί την περαιτέρω συγκεκριμενοποίηση των υπαρχόντων από την ανάλυση κλάσεων και αντικειμένων αυτών, όπως επίσης και τη δημιουργία νέων, αν αυτό χρειαστεί.

Η αντικειμενοστρεφής προσέγγιση αποδεικνύεται και εδώ επιπλέον ευέλικτη αφού, εξαιτίας της δυνατότητας που παρέχει για αφαίρεση και απόκρυψη δεδομένων, οι λεπτομέρειες κάποιων κλάσεων μπορούν να παραμείνουν «αόριστες» και να αφεθούν έτσι έως και το στάδιο της υλοποίησης. Αυτή η πρακτική όχι μόνο δεν δημιουργεί πρόβλημα στο μοντέλο σχεδίασης αλλά και δεν αποτρέπει τον σχεδιαστή να παλινδρομεί από τη σχεδίαση στην ανάλυση στην προσπάθειά του να καταλήξει στη βέλτιστη λύση. Είναι αναπόφευκτο να αντιπαραβάλλουμε αυτήν τη δυνατότητα με τη δομημένη ανάλυση και σχεδίαση, όπου μια τέτοια παλινδρόμηση είναι δύσκολη και κοστοβόρα. Στην περίπτωση που εξετάζουμε εδώ, ο σχεδιαστής μπορεί να συγκεντρωθεί στην επίλυση του προβλήματος που αντιμετωπίζει χωρίς να περιορίζεται από συγκεκριμένες υλοποιήσεις ή αρχιτεκτονικές υπολογιστών. Έτσι επιτυγχάνεται μέγιστη ευελιξία στη μοντελοποίηση, όπως επίσης και διευκολύνεται η επαναχρησιμοποίηση διαφόρων τμημάτων του σχεδιαστικού μοντέλου.

Εξαιτίας της εκφραστικής δύναμης της αντικειμενοστρεφούς τεχνολογίας, η αντικειμενοστρεφής ανάλυση δεν είναι εντελώς διακριτή από την αντικειμενοστρεφή σχεδίαση. Στην ανάλυση μοντελοποιούμε τον κόσμο του προβλήματος, ενώ στη σχεδίαση το λογισμικό που θα το επιλύσει, χρησιμοποιώντας και στις δύο περιπτώσεις τα ίδια εργαλεία: τις κλάσεις.

### Άσκηση I/Κεφάλαιο 9

Προβληματιστείτε γύρω από το εξής θέμα: η οπισθοδρόμηση από τη σχεδίαση στην ανάλυση και αντίστροφα ενδεχομένως (και αυτό είναι το πιθανότερο) να επιφέρει μεταβολές στις κλάσεις και γενικότερα στα στοιχεία λογισμικού και την τεκμηρίωση της ανάλυσης. Οι μεταβολές αυτές πρέπει καταγράφονται ξεχωριστά από τις αρχικές εκδόσεις των στοιχείων του μοντέλου ανάλυσης ή μήπως αρκεί που μετατρέπουν τις αρχικές εκδόσεις των στοιχείων αυτών σε μεταγενέστερες; Αν ισχύει το δεύτερο, δεν χάνεται το σύνορο των φάσεων της ανάπτυξης λογισμικού;

## ΕΝΟΤΗΤΑ 9.2. ΤΟ ΜΟΝΤΕΛΟ ΣΧΕΔΙΑΣΗΣ

Το μοντέλο σχεδίασης εξειδικεύει το μοντέλο ανάλυσης με σκοπό να το μεταφέρει από το πεδίο του προβλήματος στο πεδίο της επίλυσης. Κατά το στάδιο αυτό, ο αναλυτής/σχεδιαστής καλείται να αναπτύξει λεπτομερώς τα αντικείμενα που έχουν προηγουμένως αναγνωριστεί και να ορίσει τις μεταξύ τους αλληλεπιδράσεις. Γι' αυτό τον σκοπό, η ενοποιημένη προσέγγιση ανάπτυξης λογισμικού προβλέπει την απευθείας χρήση και εξειδίκευση των υπάρχοντων εργαλείων της UML, εκ των οποίων τα βασικότερα έχουν ήδη χρησιμοποιηθεί κατά την ανάλυση. Έτσι, η μετάβαση αυτή επιτυγχάνεται

με τον ευκολότερο δυνατό και φυσικό τρόπο, στα πλαίσια πάντα της αντικειμενοστρεφούς φιλοσοφίας και προσέγγισης. Αυτό συμβαίνει γιατί κατά τη σχεδίαση δεν δημιουργούμε συστατικά λογισμικού (διαγράμματα, περιγραφές) νέου τύπου, αλλά «χτίζουμε» πάνω σε ό,τι ήδη έχουμε από την ανάλυση.

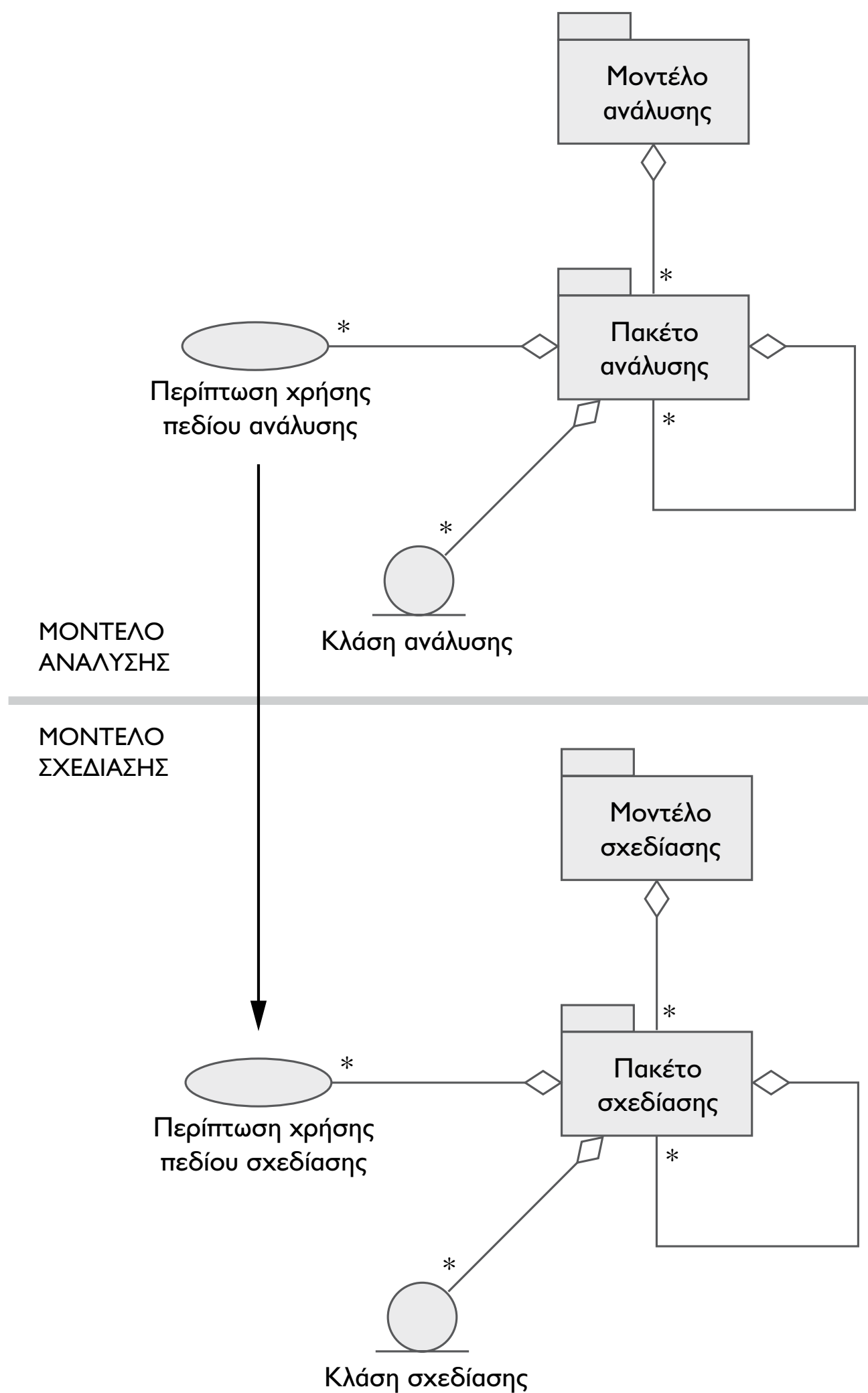
Ο στόχος της αντικειμενοστρεφούς σχεδίασης, λοιπόν, είναι η ακριβής περιγραφή των συνιστάμενων υλοποιήσιμων τμημάτων μιας λύσης λογισμικού. Μια τέτοια περιγραφή περιέχει διάφορα συστατικά λογισμικού ή τεχνουργήματα (artifacts), δηλαδή τεχνητές και υλοποιήσιμες περιγραφές των αντικειμένων του φυσικού κόσμου που εντάσσονται στο πεδίο του προβλήματος που αντιμετωπίζουμε. Η σχεδίαση εμπεριέχει επίσης περιγραφές για όλα τα επιπρόσθετα αντικείμενα που κάποια λύση μπορεί να χρειάζεται για την επιτυχή υλοποίησή της. Σε αυτή την ενότητα θα παρουσιάσουμε πώς περιγράφει η UML τα τεχνουργήματα (artifacts) που εμπλέκονται στη δημιουργία ενός μοντέλου σχεδίασης λογισμικού και πώς συνδέεται η ανάλυση με τη σχεδίαση κατά την ενοποιημένη προσέγγιση ανάπτυξης λογισμικού.

### 9.2.1. Οι περιπτώσεις χρήσης στο μοντέλο σχεδίασης

Ο εντοπισμός περιπτώσεων χρήσης και η αναλυτική καταγραφή τους είναι από τις πιο βασικές διαδικασίες κατά την ενοποιημένη προσέγγιση ανάπτυξης λογισμικού. Οι περιπτώσεις χρήσης αποτελούν μια πλήρη καταγραφή των λειτουργικών απαιτήσεων του συστήματος μέσα από το πρίσμα των διαφόρων χρηστών του, είτε είναι φυσικά πρόσωπα είτε άλλα περιφερειακά συστήματα. Επιπλέον, καθορίζουν με όσο το δυνατόν σαφή και απόλυτο τρόπο τα όρια του υπό ανάπτυξη συστήματος. Αποτελούν, έτσι, το σημείο αναφοράς του αρχικού προβλήματος προς επίλυση μεταξύ προγραμματιστών, αναλυτών/σχεδιαστών, πελατών και άλλων εμπλεκόμενων μερών, όπως με σαφήνεια φαίνεται στο Σχήμα 8.9. Εξ ορισμού, ο εντοπισμός και η προδιαγραφή των περιπτώσεων χρήσης είναι διαδικασίες που εντάσσονται πρωτίστως στη φάση της ανάλυσης, αφού βοηθούν στο να μεταφερθεί κάποιο φυσικό πρόβλημα σε ένα επίπεδο αφαίρεσης πιο κοντά στην προδιαγραφή της λύσης του λογισμικού. Για όλους αυτούς τους λόγους, οι περιπτώσεις χρήσης είναι απαραίτητες και κατά τη διαδικασία της σχεδίασης. Αποτελούν

μια σχετικά σταθερή στο χρόνο καταγραφή των λειτουργικών απαιτήσεων και των ορίων του συστήματος. Σημειώστε το «σχετικά»: η σχετικότητα εδώ δεν οφείλεται σε αδυναμία έκφρασης του αντικειμενοστρεφούς μοντέλου, αλλά στον ρυθμό μεταβολής των απαιτήσεων από το λογισμικό, ο οποίος, όπως έχουμε αναφέρει, μπορεί να είναι ταχύτερος από την ίδια την ανάπτυξη μιας εφαρμογής.

**Σχήμα 9.1** Απεικόνιση περιπτώσεων χρήσης από την ανάλυση στη σχεδίαση.

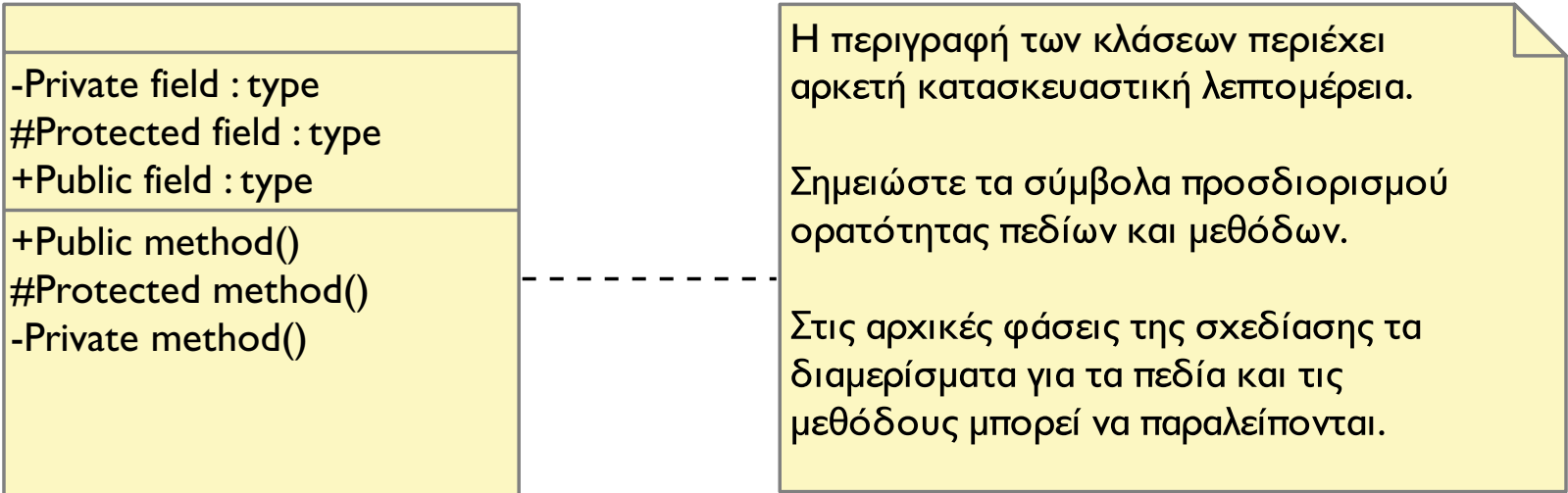


Όπως απεικονίζεται στο Σχήμα 9.1, κατ' αντιστοιχία με την περίπτωση της αντικειμενοστρεφούς ανάλυσης (Σχήμα 8.13), στο επίπεδο της σχεδίασης η κάθε περίπτωση χρήσης μεταφράζεται σε ένα ή περισσότερα πακέτα σχεδίασης. Τα πακέτα αυτά, με τη σειρά τους, αποτελούνται από μία ή περισσότερες κλάσεις σχεδίασης κ.ο.κ. Αξίζει να αναφερθεί ξανά πως τα δομικά στοιχεία του μοντέλου ανάλυσης, δηλαδή οι κλάσεις αντικειμένων και τα πακέτα ανάλυσης, πολλές φορές αντιστοιχούν αυτούσια στα δομικά στοιχεία της σχεδίασης. Με άλλα λόγια, είναι συνηθισμένο ένα πακέτο ή μία κλάση του μοντέλου ανάλυσης να αντικατοπτρίζεται ακριβώς σε ένα πακέτο ή μία κλάση αντιστοίχως του μοντέλου σχεδίασης. Είναι, βέβαια, αντιληπτό πως ο αναλυτής/σχεδιαστής τείνει να μετακινείται από το αφαιρετικό επίπεδο της ανάλυσης στο πιο πρακτικό επίπεδο της σχεδίασης. Κατά τη μετακίνηση αυτή, ο στόχος του πρέπει να είναι η όσο το δυνατόν λεπτομερής και υλοποιήσιμη καταγραφή των υπάρχόντων κλάσεων και η αλλαγή ή/και η δημιουργία καινούριων, αν αυτό κριθεί σκόπιμο. Όπως θα δούμε παρακάτω, ένας επιπλέον στόχος που θα πρέπει να επιτευχθεί είναι και η λεπτομερής περιγραφή της συνεργασίας μεταξύ κλάσεων, έτσι ώστε να ικανοποιηθούν οι περιγραφόμενες από τις περιπτώσεις χρήσης λειτουργικές απαιτήσεις του συστήματος λογισμικού.

### Συμβολισμοί UML

Στο Σχήμα 9.2 αναπαράγονται τα βασικά σύμβολα της UML για την αναπαράσταση κλάσεων και διαγραμμάτων κλάσεων. Όπως παρατηρούμε, ο συμβολισμός είναι πιο λεπτομερής στο στάδιο της σχεδίασης, μιας και σκοπός είναι να παραστήσουμε ένα επίπεδο αφαίρεσης περιπτώσεων χρήσης –και άρα ένα τμήμα του αρχικού προβλήματος– με τρόπο σαφή και υλοποιήσιμο. Έμφαση δίνεται τόσο στα πεδία και στις μεθόδους των κλάσεων όσο και στην εμβέλεια (scope) και στους τύπους τους, δηλαδή στοιχεία που κατεξοχήν αφορούν την υλοποίηση. Στο Σχήμα 9.2 φαίνονται οι τρόποι σύνδεσης και συσχετισμού στο διάγραμμα κλάσεων. Είναι κατανοητό πως ένα διάγραμμα κλάσεων μπορεί να απεικονίσει μόνο τα δομικά χαρακτηριστικά μιας περίπτωσης χρήσης και δεν περιλαμβάνει άλλες πληροφορίες, όπως χρονισμός ενεργειών.

**Σχήμα 9.2** Συμβολισμοί UML για κλάσεις.



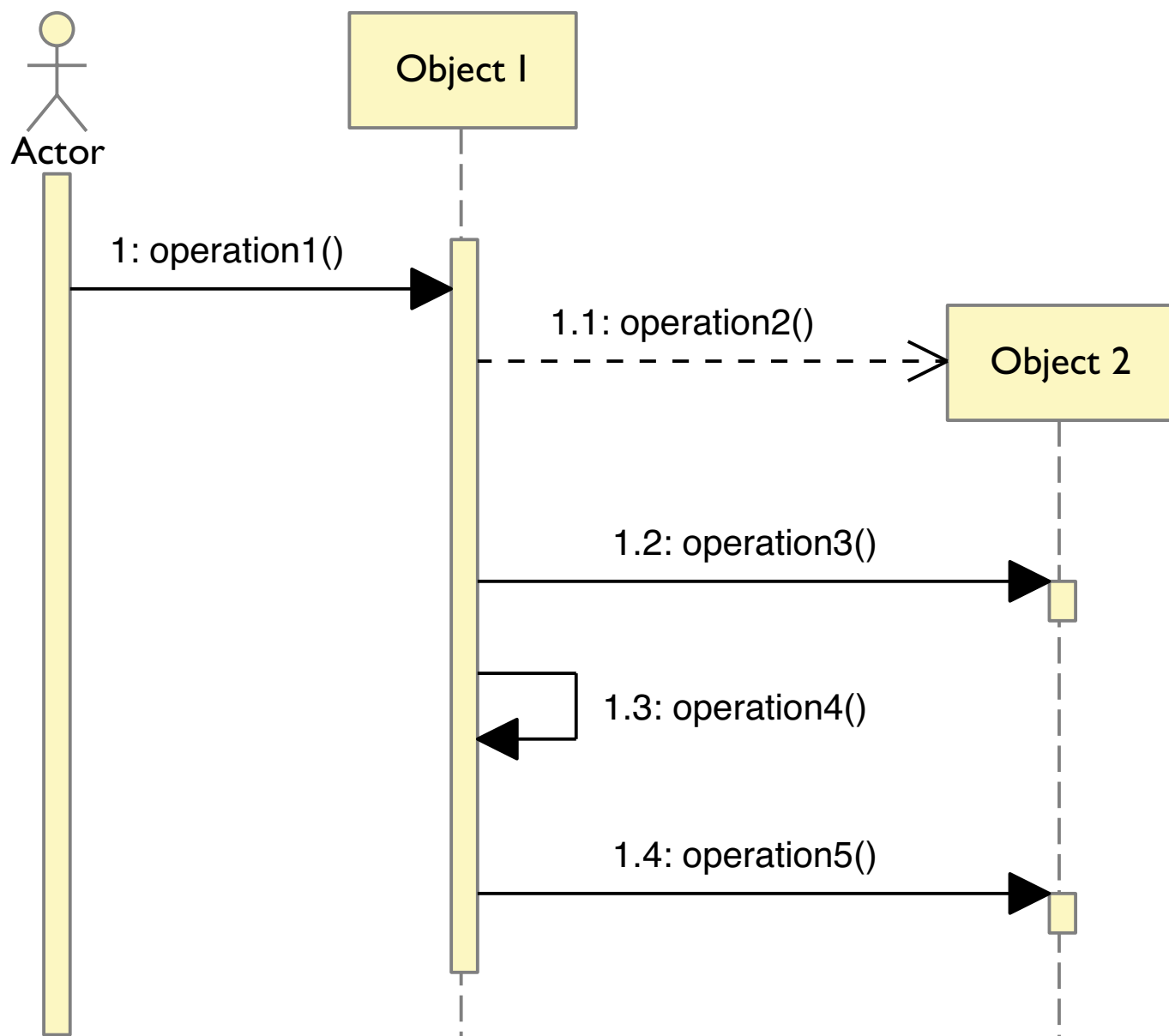


### 9.2.2. Πώς σχεδιάζεται η περίπτωση χρήσης

Όπως φαίνεται στο Σχήμα 9.1, η κάθε περίπτωση χρήσης, αφού αναλυθεί, μπορεί να περιγραφεί από μια σειρά διεργασιών μεταξύ ενός συνόλου αντικειμένων. Αφού τα αντικείμενα ανήκουν στις αντίστοιχες προδιαγεγραμμένες κλάσεις τους, η UML μάς δίνει τη δυνατότητα να περιγράψουμε τον τρόπο με τον οποίο αυτά επικοινωνούν με δύο συμπληρωματικούς μηχανισμούς: τα διαγράμματα κλάσεων και τα διαγράμματα αλληλεπιδράσεων. Τα διαγράμματα κλάσεων αποτελούν μία αρχιτεκτονική όψη για τις διάφορες περιπτώσεις χρήσης. Τα διαγράμματα αλληλεπιδράσεων χωρίζονται σε διαγράμματα αλληλουχίας ή σειράς ή ακολουθίας (sequence) και σε διαγράμματα συνεργασίας (collaboration), και απεικονίζουν με ποιο τρόπο και μέσω ποιων μεθόδων επικοινωνούν τα διάφορα αντικείμενα για να φέρουν εις πέρας την περίπτωση χρήσης. Με τη χρήση των δύο αυτών μηχανισμών της UML, ο αναλυτής/σχεδιαστής επιτυγχάνει, πλέον, την πλήρη προδιαγραφή μιας περίπτωσης χρήσης με τρόπο που την καθιστά, θεωρητικά τουλάχιστον, κατανοητή και συγχρόνως ευθέως υλοποιήσιμη. Τα διαγράμματα συνεργασίας αναφέρθηκαν στο Κεφάλαιο 8, ενώ τα διαγράμματα ακολουθίας ακολουθούν τον συμβολισμό όπως στο Σχήμα 9.3.



### Σχήμα 9.3 Συμβολισμοί UML



Στα διαγράμματα σειράς ή ακολουθίας (sequence) ο χειριστής αντιπροσωπεύεται από ένα αντικείμενο.

Κάθε περίπτωση χρήσης περιγράφεται από μια ακολουθία κλήσεων μεταξύ αντικειμένων που συμμετέχουν στην υλοποίησή της.

Κάθε αντικείμενο έχει μια "γραμμή ζωής" (lifeline) η οποία παριστάνεται με τη διακεκομμένη γραμμή στο σχήμα. Όταν το αντικείμενο είναι ενεργό, τότε η διακεκομμένη γραμμή μετατρέπεται σε πλαίσιο. Οι κλήσεις μεθόδων δηλώνονται με ένα βέλος, στο οποίο σημειώνεται το όνομα και η σειρά κλήσης της μεθόδου που καλείται.

### 9.2.3. Διεπαφές στο μοντέλο σχεδίασης

Οι διεπαφές (interfaces) αποτελούν έναν κεντρικό μηχανισμό στην αντικειμενοστρεφή φιλοσοφία. Μέσω αυτών ο σχεδιαστής ορίζει τρόπους επικοινωνίας μεταξύ των διαφόρων αντικειμένων και τεχνουργημάτων του μοντέλου σχεδίασης. Ήδη από το Κεφάλαιο 8 έχουμε δει ότι οι κλάσεις μπορούν να διαχωριστούν σε συνοριακές, οντοτήτων και ελέγχου. Από αυτές, οι συνοριακές κλάσεις αποτελούν, εξ ορισμού, το πρωτόκολλο επικοινωνίας μεταξύ του υπό σχεδίαση συστήματος και του έξω κόσμου, δηλαδή των περιφερειακών συστημάτων, συσκευών κ.λπ. Υπάρχουν όμως και άλλες εκφάνσεις διεπαφών που μπορούν να οριστούν ρητά τόσο στη UML όσο και σε διάφορες αντικειμενοστρεφείς γλώσσες προγραμματισμού και αφορούν αντικείμενα του ίδιου συστήματος. Όμως, πριν προχωρήσουμε παρακάτω, θα ήταν χρήσιμο να ορίσουμε πώς εννοούμε τη διεπαφή στην περίπτωση της αντικειμενοστρεφούς τεχνολογίας.

#### **Διεπαφή (interface):**

Η διεπαφή μιας κλάσης αντικειμένων είναι το σύνολο των πεδίων και μεθόδων της κλάσης που μπορούν να αναγνωστούν, να μετατραπούν ή να κληθούν από κάποιο αντικείμενο της ίδιας ή άλλης κλάσης.

Με άλλα λόγια, η διεπαφή αποτελεί ένα «δημόσια κοινοποιημένο» πρωτόκολλο επικοινωνίας, με την ευρεία έννοια του όρου, ή αλλιώς ένα σύνολο επιθυμητών τρόπων συναλλαγής των αντικειμένων μιας κλάσης με το περιβάλλον τους.

Από τον ορισμό αυτό συμπεραίνουμε πως ο ακριβής ρόλος και έκφραση μιας διεπαφής θα εξαρτηθεί τόσο από τον ορισμό της προγραμματιστικά όσο και από τη σχέση μεταξύ των εμπλεκόμενων κλάσεων. Για παράδειγμα, είναι κατανοητό πως η διεπαφή μιας κλάσης, όσον αφορά τα αντικείμενα της ίδιας της κλάσης, είναι το σύνολο των πεδίων και των μεθόδων της ανεξαρτήτως από το αν έχουν οριστεί ως κοινοποιημένα ή κρυφά. Μία

μέθοδος της κλάσης θα έχει πρόσβαση σε όλα τα πεδία και τις μεθόδους που έχουν οριστεί μέσα σε αυτή. Για ένα αντικείμενο μιας δεύτερης κλάσης όμως, αναλόγως με τη σχεδίαση που έχουμε, όπως αυτή αποτυπώνεται στο διάγραμμα κλάσεων (βλ. Σχήμα 9.2), η επικοινωνία θα περιοριστεί στα κοινοποιημένα (public) πεδία και μεθόδους της πρώτης κλάσης.

## Παράδειγμα

Ας επιστρέψουμε στην εφαρμογή «Επίκουρος». Σε αυτή την εφαρμογή, διάφορες κλάσεις οντοτήτων περιέχουν ευαίσθητα δεδομένα όπως, για παράδειγμα, τον αριθμό ταυτότητας των καθηγητών στην κλάση «καθηγητής» κ.ο.κ. Αν απαγορεύσουμε την απευθείας αλλαγή των αντίστοιχων πεδίων και αντ' αυτού ορίσουμε πως οι όποιες αλλαγές θα γίνονται μόνο μέσα από συγκεκριμένες μεθόδους, επιτυγχάνουμε μεγαλύτερο έλεγχο σχετικά με τον τρόπο που μπορούν τέτοια στοιχεία να αλλαχθούν. Το σύνολο αυτών των μεθόδων στην ουσία θα αποτελέσουν μία διεπαφή μεταξύ της κλάσης «καθηγητής» και του υπόλοιπου συστήματος.

Οι μέθοδοι αυτές, αν και ίσως ακατανόητες στην αρχή της καριέρας ενός σχεδιαστή λογισμικού, είναι πολύ χρήσιμες: επιτρέπουν την πραγματοποίηση μεταβολών στην υλοποίηση της κλάσης χωρίς να αλλάξει τίποτε στον τρόπο που «οι άλλοι» βλέπουν την κλάση, πράγμα που οδηγεί στην κατασκευή θωρακισμένου και επαναχρησιμοποιήσιμου λογισμικού. Σε αρκετά βιβλία προγραμματισμού αναφέρονται ως «accessors», από το ρήμα access, που σημαίνει «προσπελαύνω, αποκτώ πρόσβαση σε κάτι».

### Δραστηριότητα I/Κεφάλαιο 9

Υλοποιήστε ένα interface για τη λειτουργία «εκτύπωση βαθμολογίας σπουδαστή» του λογισμικού «Επίκουρος», σχολιάζοντας κατάλληλα τη χρησιμότητα και τον ρόλο του.

#### 9.2.4. Κλάσεις στο μοντέλο σχεδίασης

Όπως έχουμε δει, οι κλάσεις κατέχουν κεντρικό ρόλο στη σχεδίαση, και προέρχονται από τις κλάσεις που έχουν οριστεί στο μοντέλο ανάλυσης. Κατά τη σχεδίαση, οι κλάσεις αυτές πρέπει να καθοριστούν με λεπτομερή τρόπο, έτσι ώστε το μοντέλο να μπορέσει να ικανοποιήσει τις λειτουργικές ανάγκες του προς ανάπτυξη συστήματος. Γι' αυτό τον σκοπό, ο σχεδιαστής έχει στη διάθεσή του τα εργαλεία που έχουμε προαναφέρει όσον αφορά τις κλάσεις του συστήματος:

- Τα διαγράμματα κλάσεων.
- Τα διαγράμματα αλληλεπιδράσεων στα οποία ανήκουν:
  - τα διαγράμματα αλληλουχίας ή ακολουθίας ή σειράς (sequence) και
  - τα διαγράμματα συνεργασίας (collaboration).

Τα μεν διαγράμματα κλάσεων παρέχουν μια στατική απεικόνιση της αρχιτεκτονικής που θα πρέπει να υλοποιηθεί, τα δε διαγράμματα αλληλεπιδράσεων βοηθούν στην συγκεκριμενοποίηση και περαιτέρω κατανόηση των λειτουργικών απαιτήσεων του προβλήματος μέσα από τη δυναμική απεικόνιση των περιπτώσεων χρήσης, η οποία περιέχει και τη διάσταση του χρόνου. Τα διαγράμματα κλάσεων αποτυπώνουν τις εμπλεκόμενες κλάσεις και τις συσχετίσεις τους σε αρχιτεκτονικό επίπεδο. Από την άλλη, τα διαγράμματα ακολουθίας επικεντρώνονται στη σειρά με την οποία ανταλλάσσονται τα μηνύματα μεταξύ των κλάσεων (δηλαδή οι κλήσεις των μεθόδων των διεπαφών τους από άλλες κλάσεις), ενώ τα διαγράμματα συνεργασίας επικεντρώνονται στις σχέσεις μεταξύ των αντικειμένων και είναι ικανά να αποτυπώσουν τη συνεργασία μεταξύ πολλών κλάσεων.

Ενώ τα δύο αυτά εργαλεία (τα διαγράμματα κλάσεων και τα διαγράμματα αλληλεπιδράσεων) είναι εξίσου σημαντικά για την υλοποίηση του συστήματος, πολλές φορές οι σχεδιαστές δίνουν μεγαλύτερη βαρύτητα στα διαγράμματα κλάσεων εις βάρος των διαγραμμάτων αλληλεπιδράσεων. Αυτό μπορεί να συμβαίνει για δύο λόγους: για ένα σύστημα λογισμικού, τα διαγράμματα κλάσεων είναι λίγα –συνήθως ένα για κάθε υποσύστημα. Από την

άλλη, τα διαγράμματα αλληλεπιδράσεων είναι πάρα πολλά, αναλόγως με τον αριθμό των περιπτώσεων χρήσης και των λειτουργικών απαιτήσεων του συστήματος. Μάλιστα, για κάθε περίπτωση χρήσης μπορούν να υπάρχουν εναλλακτικές ροές (βλ. Ενότητα 8.5.2) οι οποίες πολλαπλασιάζουν τον αριθμό των διαγραμμάτων ακολουθίας. Συνεπώς, είναι πολύ πιο εύκολο να δημιουργήσει και να συντηρήσει κάποιος στατικά διαγράμματα κλάσεων παρά διαγράμματα αλληλεπιδράσεων.

Ένας επιπρόσθετος λόγος για αυτή την άτυπη προτίμηση είναι ότι τα διαγράμματα κλάσεων μπορούν να αναγνωστούν και να χρησιμοποιηθούν ευκολότερα από προγραμματιστές και έτσι βοηθούν στην υλοποίηση πρωτότυπων συστημάτων σε σύντομο χρόνο. Αυτά τα πλεονεκτήματα όμως είναι πλασματικά και οι σχεδιαστές που κλίνουν προς τη χρήση μόνο διαγραμμάτων κλάσεων δεν λαμβάνουν υπόψη τους την επαναληπτική και επαυξητική φύση της αντικειμενοστρεφούς προσέγγισης και αφαιρούν από την εκφραστική της δύναμη. Δεν αρκούν μόνο τα διαγράμματα κλάσεων για να γίνει μια επιτυχής υλοποίηση που να πληροί τις προδιαγραφές της εφαρμογής. Πρέπει τα στατικά και τα δυναμικά διαγράμματα να χρησιμοποιηθούν επαναλαμβανόμενα, τροφοδοτώντας τα μεν τα δε μέχρις ότου συγκλίνουν σε μία αποδεκτή λύση. Στο παράδειγμα που ακολουθεί γίνεται φανερό πώς τα διαγράμματα κλάσεων και τα διαγράμματα ακολουθίας και συνεργασίας αλληλοσυμπληρώνονται και πώς η επαναληπτική, εκ περιτροπής εξειδίκευσή τους μπορεί να μας οδηγήσει σε μια πιο ορθολογική και υλοποιήσιμη λύση.

## Δραστηριότητα 2/Κεφάλαιο 9

Υλοποιήστε ένα interface για τη λειτουργία «εκτύπωση βαθμολογίας σπουδαστή» του λογισμικού «Επίκουρος», σχολιάζοντας κατάλληλα τη χρησιμότητα και τον ρόλο του.

Ας αφήσουμε προσωρινά την εφαρμογή «Επίκουρος» και ας ξαναδούμε την εφαρμογή συναρμολόγησης του αεροσκάφους του παραδείγματος του Κεφαλαίου 7. Ας υποθέσουμε πως θέλουμε να αναπτύξουμε ένα λογισμικό εξομοιωτή πτήσεων και, πιο συγκεκριμένα, τον αυτόματο πιλότο. Από τις διάφορες λειτουργίες που μπορεί να κάνει ένας αυτόματος πιλότος, ας επικεντρωθούμε στην περίπτωση χρήσης της απογείωσης. Τροποποιήστε κατάλληλα το διάγραμμα κλάσεων του ίδιου παραδείγματος και κατασκευάστε ένα διάγραμμα συνεργασίας.

### 9.2.5. Αρχιτεκτονική όψη και υποσυστήματα

Είναι σύνηθες και απολύτως απαραίτητο για τα μεγάλα συστήματα λογισμικού να χωρίζονται σε μικρότερα συνιστώμενα τμήματα ή υποσυστήματα. Αυτό πρακτικά εφαρμόζει την αρχή του «διαίρει και βασίλευε» στην ανάπτυξη λογισμικού, με όλα τα πλεονεκτήματα μιας τέτοιας μεθοδολογίας, όπως έχει ήδη αναφερθεί.

Μερικά από αυτά τα πλεονεκτήματα είναι τα παρακάτω:

- Το πρόβλημα επιλύεται ταχύτερα και πιο οργανωμένα όταν επικεντρωνόμαστε σε μικρότερα προβλήματα παρά σε μεγαλύτερα.
- Κάνει δυνατό τον διαχωρισμό της εργασίας σε ανεξάρτητες ομάδες αναλυτών, σχεδιαστών, προγραμματιστών κ.λπ., ανάλογα με το υποσύστημα και τις λειτουργικές και μη απαιτήσεις του.
- Ενθαρρύνει την επαναχρησιμοποίηση υποσυστημάτων σε άλλα εγχειρήματα που μπορεί να προκύψουν.
- Μπορεί να προκύψει φυσικά από το πρόβλημα και την έκφρασή του στον πραγματικό κόσμο.



Αυτά είναι μερικά μόνο από τα πλεονεκτήματα αυτής της τακτικής και η επίτευξή τους προϋποθέτει την εφαρμογή κάποιων παραδοχών. Αυτές οι παραδοχές θα γίνουν ευδιάκριτες μέσα από τον ορισμό του υποσυστήματος και του συστατικού στοιχείου.

**Υποσύστημα** είναι ένα λειτουργικό τμήμα ενός ολοκληρωμένου συστήματος λογισμικού το οποίο, μέσα από μια σειρά εννοιολογικά συναφών λειτουργιών που περιέχει και υλοποιεί, παρέχει υπηρεσίες στη συνολική λύση που υλοποιεί το ολοκληρωμένο σύστημα.

Το συστατικό στοιχείο (component) της UML είναι πολύ σχετικό με το υποσύστημα. Η πλέον συνηθισμένη προσέγγιση είναι πως το συστατικό στοιχείο αποτελεί την υλοποίηση ενός υποσυστήματος. Άρα, ενώ και τα δύο αποτελούν μέρη της αρχιτεκτονικής, είναι συνηθέστερο η αρχιτεκτονική κατά την ανάλυση να παρουσιάζεται βάσει υποσυστημάτων, ενώ η αρχιτεκτονική κατά την υλοποίηση να παρουσιάζεται βάσει συστατικών στοιχείων του λογισμικού. Η πιο πρόσφατη έκδοση της UML, η UML 2, δεν είναι σαφής ως προς τον διαχωρισμό των συστατικών στοιχείων και των υποσυστημάτων καθώς δεν διαχωρίζει τις δύο αυτές δομικές οντότητες σε σχέση με τη χρήση τους κατά τη μοντελοποίηση. Μάλιστα, υποστηρίζει πως η χρήση του ενός έναντι του άλλου επαφίεται στη μεθοδολογία που επιλέγει να ακολουθήσει ο αναλυτής/σχεδιαστής. Προκειμένου να διατηρήσουμε τα δύο αυτά στοιχεία διακριτά, θα ακολουθήσουμε την προσέγγιση που αναφέρουμε και πιο πάνω: κατά τη σχεδίαση θα χρησιμοποιούμε υποσυστήματα και κατά την υλοποίηση θα χρησιμοποιούμε συστατικά στοιχεία λογισμικού, αποδεχόμενοι την παραδοχή ότι στη γενική περίπτωση ένα υποσύστημα αποτελείται από περισσότερα του ενός συστατικά στοιχεία.

Η αρχιτεκτονική όψη ή σχεδίαση συνήθως καθορίζεται νωρίς κατά την ανάλυση και σχεδίαση ενός συστήματος. Όπως και οι υπόλοιπες διαδικασίες της ενοποιημένης προσέγγισης ανάπτυξης λογισμικού, έχει επαναληπτικό και επαυξητικό χαρακτήρα. Έτσι, είναι σύνηθες να ξεκινούμε από την αρχιτεκτονική όψη, να συνεχίζουμε με τη σχεδίαση των επιμέρους υποσυστημάτων βάσει της οποίας στη συνέχεια αναπροσδιορίζουμε την αρχιτεκτονική

όψη κ.ο.κ. Ο ακριβής τρόπος και η σειρά με την οποία θα χρειαστεί να ακολουθήσουμε αυτά τα βήματα δεν είναι δυνατό να είναι γνωστά εκ των προτέρων. Θα εξαρτηθούν από τη φύση του προβλήματος, από το μέγεθός του, από τη μεθοδολογία που προτιμά να ακολουθεί ο εκάστοτε αναλυτής/σχεδιαστής, όπως και από πολλές άλλες παραμέτρους. Σε κάθε περίπτωση όμως, πρέπει να τηρούνται τα παρακάτω:

- Το κάθε υποσύστημα θα πρέπει να έχει διακριτό ρόλο στη συνολική αρχιτεκτονική του συστήματος, δηλαδή τα επιμέρους τμήματά του θα πρέπει να εξυπηρετούν κάποιο συγκεκριμένο σκοπό, να καλύπτουν έναν αριθμό συναφών περιπτώσεων χρήσης ή λειτουργικών απαιτήσεων με ακρίβεια και συνοχή.
- Η επικοινωνία και η διαλειτουργικότητα μεταξύ των υποσυστημάτων θα πρέπει να είναι ορισμένη με τρόπο όσο το δυνατόν πιο ακριβή. Αυτό επιτυγχάνεται με την εισαγωγή διεπαφών μεταξύ των υποσυστημάτων, οι οποίες, όπως έχουμε δει, είναι ανεξάρτητες από τις λεπτομέρειες της οποιασδήποτε υλοποίησης.
- Το μοντέλο ελέγχου μεταξύ των υποσυστημάτων του λογισμικού πρέπει να οριστεί. Με αυτό εννοούμε τον τρόπο με τον οποίο θα συντονίζονται τα υποσυστήματα. Για παράδειγμα, μπορεί να έχουμε κεντρικό έλεγχο ή ασύγχρονο έλεγχο. Η επιλογή του μοντέλου ελέγχου θα εξαρτηθεί από τις λειτουργικές απαιτήσεις του συστήματος ή θα είναι στις μη λειτουργικές απαιτήσεις βάσει συμβολαίου.
- Το κάθε υποσύστημα θα πρέπει να αναλυθεί ανεξάρτητα στα δομικά του μέρη και αυτά με τη σειρά τους στα δικά τους, με τα εργαλεία που έχουμε περιγράψει στις προηγούμενες ενότητες. Όπως έχουμε τονίσει επανειλημμένως, οι παλινδρομήσεις είναι αναμενόμενες και αναγκαίες μεταξύ των διαφόρων επιπέδων της σχεδίασης. Η περαιτέρω ανάλυση και σχεδίαση είναι αναμενόμενο να ρίξει φως σε πτυχές του προβλήματος που ίσως επηρεάσουν την αρχική αρχιτεκτονική ή τις περιπτώσεις χρήσης ή τις διεπαφές μεταξύ κλάσεων ή υποσυστημάτων που έχουν οριστεί κ.ο.κ.



## Δραστηριότητα 3/Κεφάλαιο 9

Ας επιστρέψουμε στην εφαρμογή «Επίκουρος». Οι λειτουργικές απαιτήσεις της εφαρμογής περιγράφονται στο Σχήμα 8.11 και οι περιπτώσεις χρήσης περιγράφονται στο Σχήμα 8.17 του προηγούμενου κεφαλαίου. Μπορείτε να ορίσετε υποσυστήματα;

### 9.2.6. Μοντέλο διάταξης (Deployment)

Όπως είδαμε στην προηγούμενη ενότητα, η αρχιτεκτονική όψη μέσω της μοντελοποίησης υποσυστημάτων και συστατικών στοιχείων μπορεί να μας δώσει μια ολοκληρωμένη εικόνα των εμπλεκόμενων, λογικών μερών ενός συστήματος. Ενώ η αρχιτεκτονική όψη είναι πολύ χρήσιμη τόσο σε αναλυτές/σχεδιαστές όσο και σε προγραμματιστές, δεν μας βοηθά να κατανοήσουμε και να περιγράψουμε τις μη λειτουργικές ανάγκες του συστήματος και τις αλληλεξαρτήσεις των διαφόρων υποσυστημάτων βάσει αυτών. Μια τέτοια απεικόνιση επιτυγχάνεται μέσω της δημιουργίας και συντήρησης ενός μοντέλου διάταξης ή, ισοδύναμα, εγκατάστασης. Το μοντέλο διάταξης απεικονίζει τον τρόπο που διατάσσονται τα διάφορα λογικά μέρη στο υλικό που έχουμε στη διάθεσή μας. Απεικονίζει, επίσης, και τις διάφορες αλληλεξαρτήσεις τους σε επίπεδο σχεδίασης, υλοποίησης και εγκατάστασης –εξού και ο όρος «μοντέλο εγκατάστασης» που συναντάται συχνά στη βιβλιογραφία και αναφέρεται στο ίδιο μοντέλο. Μιας και όταν κληθούμε να παραδώσουμε το σύστημα, αυτό δεν θα γίνει με βάση τα διαγράμματα κλάσεων και τα άλλα αφαιρετικά σχήματα που ενδεχομένως έχουμε χρησιμοποιήσει αλλά με βάση τα παράγωγα της υλοποίησης, είναι αναμενόμενο πως το μοντέλο διάταξης θα αποτελείται από αναπαραστάσεις αυτών. Έτσι, σε ένα τέτοιο μοντέλο βρίσκουμε συστατικά στοιχεία (όπως τα έχουμε παρουσιάσει και στην προηγούμενη ενότητα), στοιχεία υλικού και τους μεταξύ τους συνδέσμους.

*Σημείωση: Τα στοιχεία που απαρτίζουν το μοντέλο διάταξης ή εγκατάστασης ανήκουν στη φάση της υλοποίησης, μιας και συναντάμε βιβλιοθήκες, εκτελέσιμα αρχεία και υλικό, όπως σταθμούς εργασίας, εξυπηρετητές (servers), δικτυακές*

συνδεσμολογίες κ.λπ. Όμως, όπως και άλλα μοντέλα της ενοποιημένης προσέγγισης ανάπτυξης λογισμικού, έτσι και το μοντέλο διάταξης δεν έχει στενά όρια χρήσης. Μιας και σε κάποιο δεδομένο έργο οι μη λειτουργικές απαιτήσεις είναι γνωστές από τα πρώτα στάδια της ανάλυσής του, μοντέλα διάταξης είναι καλό να δημιουργούνται από το στάδιο της σχεδίασης (ή ακόμα νωρίτερα), έτσι ώστε να αποτελούν ένα σημείο αναφοράς μεταξύ της πιο αφαιρετικής σχεδίασης και της απολύτως πρακτικής υλοποίησης. Αυτός είναι ο λόγος για τον οποίο η περιγραφή αυτού του μοντέλου βρίσκεται σε αυτό το κεφάλαιο και όχι στο επόμενο που πραγματεύεται την υλοποίηση.

Στο μοντέλο διάταξης τα συστατικά στοιχεία είναι συνήθως συνδεδεμένα με αντίστοιχα υποσυστήματα. Για παράδειγμα, ένα υποσύστημα Α, το οποίο περιγράφεται από μία σειρά διαγραμμάτων κλάσεων και άλλων σχεδιαστικών περιγραφικών κειμένων, στην υλοποιημένη του μορφή θα αποτελέσει ένα συστατικό στοιχείο. Ανάλογα με τις προδιαγεγραμμένες λειτουργίες που θα πρέπει να προσφέρει στο σύστημα, μπορεί να είναι κάποιο εκτελέσιμο, μία βιβλιοθήκη, μία φόρμα εισόδου/εξόδου για χρήστες κ.λπ. Όταν εισάγουμε ένα συστατικό στοιχείο στο διάγραμμα διάταξης, η UML επιτρέπει τη χρήση στερεοτύπων (stereotypes) για τον προσδιορισμό του. Επίσης, οι σχέσεις μεταξύ των συστατικών στοιχείων αποδίδονται με συνδέσμους οι οποίοι μπορούν επίσης να φέρουν στερεότυπα, υποδηλώνοντας τους τρόπους εξάρτησης των συστατικών στοιχείων. Ένα τελευταίο χαρακτηριστικό των συστατικών στοιχείων είναι πως, όπως και οι κλάσεις, έτσι και αυτά, μπορούν να υλοποιούν διεπαφές. Στην περίπτωση αυτή, αν η σχέση κάποιου άλλου συστατικού στοιχείου εξαρτάται από τις λειτουργίες που προδιαγράφονται από τη διεπαφή, αυτή καταλήγει στη διεπαφή και όχι απευθείας στο αρχικό συστατικό στοιχείο. Έτσι, μπορούμε να ξεχωρίσουμε ποια συστατικά στοιχεία τηρούν μία διεπαφή (δηλαδή ακολουθούν το πρωτόκολλο επικοινωνίας που αυτή περιγράφει) και ποια όχι. Ο αναγνώστης μπορεί να βρει περισσότερες πληροφορίες στους συμβολισμούς UML που ακολουθούν.

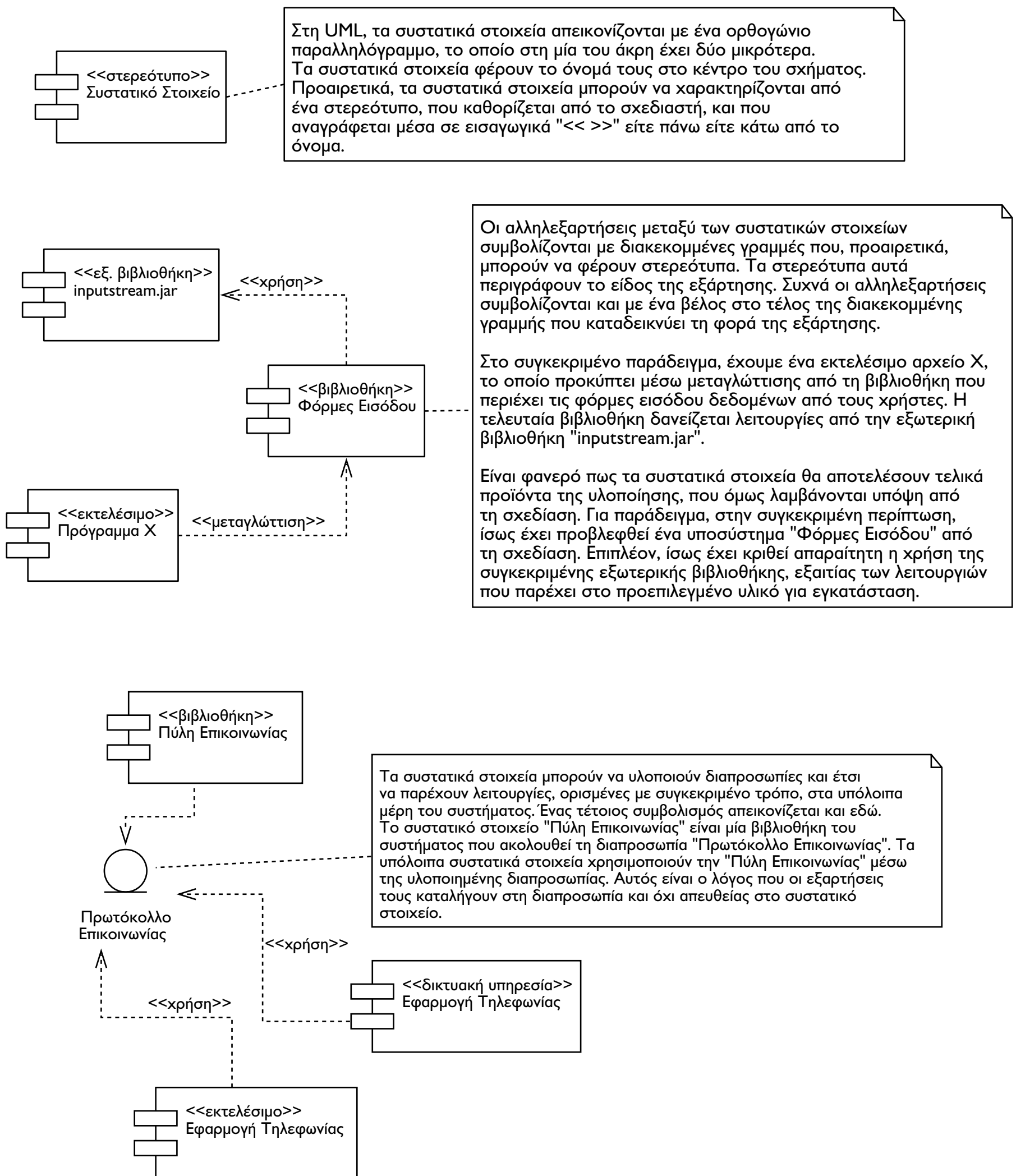
Έχοντας παρουσιάσει τη χρησιμότητα και τους συμβολισμούς των συστατικών στοιχείων, μένει να περάσουμε στη διάταξη των συστατικών στοιχείων στα διάφορα μέρη του υλικού που έχουμε στη διάθεσή μας, δηλαδή να

δημιουργήσουμε το μοντέλο διάταξης ή εγκατάστασης για το σύστημά μας. Πιο συγκεκριμένα, το διάγραμμα διάταξης ή εγκατάστασης απεικονίζει τα εξής:

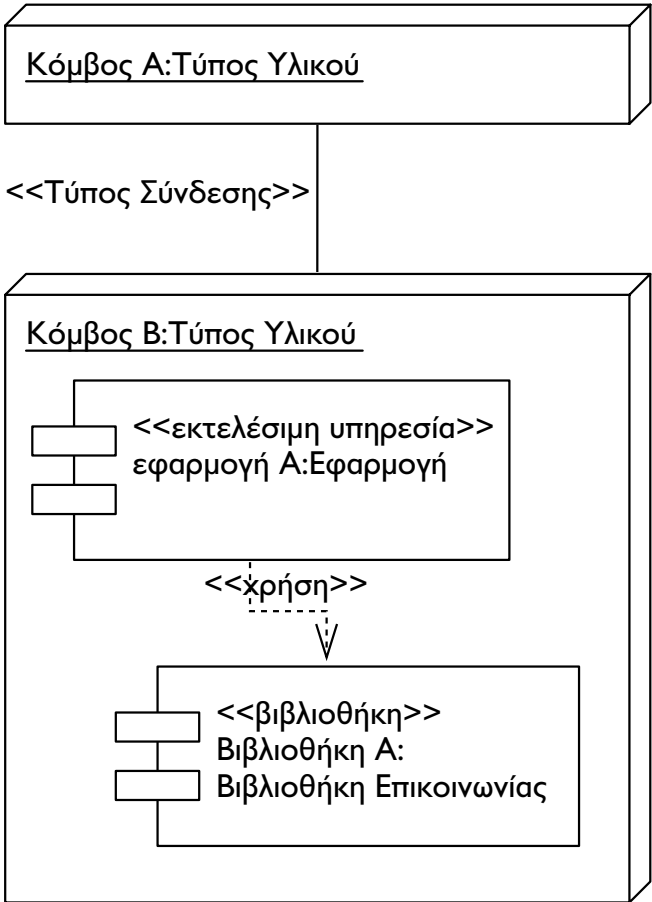
- Τους συνδέσμους επικοινωνίας μεταξύ των διαφόρων μερών του υλικού, όπως μηχανήματα, διακομιστές, εκτυπωτές κ.λπ.
- Τις σχέσεις των συστατικών στοιχείων, όπως τα έχουμε ορίσει παραπάνω, και των διαφόρων μερών του υλικού. Με άλλα λόγια, σε ποιο τμήμα του υλικού υπάρχει και εκτελείται κάθε συστατικό στοιχείο.

Στη συνέχεια οι όροι «μοντέλο διάταξης» και «μοντέλο εγκατάστασης» θα χρησιμοποιούνται ισοδύναμα. Στο μοντέλο διάταξης απεικονίζονται τόσο τα μέρη του υλικού όσο και τα συστατικά στοιχεία της τελικής εφαρμογής λογισμικού. Έχοντας ήδη καλύψει τον συμβολισμό και τη σημασία των συστατικών στοιχείων, μένει να περάσουμε και στη μοντελοποίηση του υλικού. Στο συγκεκριμένο μοντέλο, η διάταξη του υλικού έχει τη μορφή κόμβων και συνδέσμων. Οι κόμβοι συμβολίζουν τα διάφορα εμπλεκόμενα μέρη του υλικού, ενώ οι σύνδεσμοι συμβολίζουν τα διάφορα κανάλια επικοινωνίας μεταξύ των μερών του υλικού. Μέσα στους κόμβους του υλικού υπάρχουν και εκτελούνται (τρέχουν) τα συστατικά στοιχεία που έχουμε προκαθορίσει. Άρα, το μοντέλο διάταξης μας δίνει μια εικόνα του συστήματος κατά την εκτέλεση ή τη χρήση του. Ο ακριβής συμβολισμός UML δίνεται παρακάτω.

## ΣΥΜΒΟΛΙΣΜΟΙ UML (ΣΥΣΤΑΤΙΚΩΝ ΣΤΟΙΧΕΙΩΝ)



# ΣΥΜΒΟΛΙΣΜΟΙ UML (ΣΥΜΒΟΛΙΣΜΟΙ ΜΟΝΤΕΛΟΥ ΔΙΑΤΑΞΗΣ)



Σύμφωνα με το παραταξιακό μοντέλο, οι κόμβοι υλικού συμβολίζονται με κυβοειδή σχήματα. Φέρουν ένα όνομα μαζί με τον τύπο του υλικού που έχει προδιαγραφεί στις απαιτήσεις του συστήματος. Παρατηρούμε πως τόσο το υλικό όσο και τα συστατικά στοιχεία που εκτελούνται σε αυτό φέρουν μία έκφραση μαζί με έναν τύπο (ή κλάση). Αυτό συμβολίζεται όπως και στην περίπτωση των αντικειμένων και των κλάσεων, δηλαδή με τη μορφή όνομα:Τύπος. Ο διαχωρισμός αυτός είναι χρήσιμος αφού απεικονίζουμε την κατάσταση εκτέλεσης της εφαρμογής, κατά την οποία μπορεί να έχουμε διαφορετικές εκφάνσεις του ίδιου τύπου υλικού και συστατικών στοιχείων.

Το γεγονός ότι το μοντέλο διάταξης απεικονίζει την αρχιτεκτονική του συστήματος κατά την εκτέλεσή του μας αναγκάζει να προσέξουμε ένα επιπλέον χαρακτηριστικό των συστατικών στοιχείων και των κόμβων υλικού. Στα δύο αυτά σύμβολα είμαστε, πλέον, υποχρεωμένοι να εισάγουμε την έννοια της έκφανσης κατά την εκτέλεση (instantiation) σε αντιπαράθεση με τους τύπους (ή τα είδη) των τμημάτων υλικού ή των συστατικών στοιχείων. Αυτές οι έννοιες βρίσκονται σε ευθεία αναλογία (εννοιολογικά και συμβολικά) με τις έννοιες των αντικειμένων και των κλάσεων που έχουμε συναντήσει σε διάφορα σημεία του βιβλίου. Για να κατανοήσουμε τη διαφορά μεταξύ ενός είδους και μιας έκφανσής του ίσως βοηθήσει το να σκεφτόμαστε την έκφανση ως κάτι συγκεκριμένο και από και το είδος ως κάτι περιγραφικό και αφαιρετικό. Για παράδειγμα, η έννοια του ακέραιου αριθμού μπορεί να είναι μία κλάση, ενώ ο αριθμός 1 είναι μία έκφανσή της. Σε αντιστοιχία με το μοντέλο διάταξης, μπορεί να έχουμε μία κλάση (ή ένα είδος) «προσωπικός υπολογιστής» που να περιγράφει όλους τους προσωπικούς υπολογιστές, αλλά να χρειαστεί να εγκαταστήσουμε διαφορετικά συστατικά στοιχεία (για την ακρίβεια εκφάνσεις αυτών) σε συγκεκριμένους, διακριτούς υπολογιστές Κ.Ο.Κ.



## Δραστηριότητα 4/Κεφάλαιο 9

Ας αφήσουμε τη φαντασία μας ελεύθερη και ας υποθέσουμε πως χρειάζεται να αναπτύξουμε μία εφαρμογή διαδικτυακής τηλεφωνίας (VoIP) όπως, λόγου χάρη, το Skype. Τέτοιου είδους εφαρμογές επιτρέπουν στους χρήστες τους να συνομιλούν όχι μέσω του τηλεφωνικού δικτύου αλλά μέσω του διαδικτύου, μέσω πακέτων δεδομένων. Η χρήση τέτοιων προγραμμάτων απαιτεί έναν υπολογιστή με σύνδεση στο διαδίκτυο, ακουστικά και μικρόφωνο. Βιωματικά όλοι γνωρίζουμε ότι απαιτείται και η ύπαρξη κάποιου κεντρικού σημείου αναφοράς της υπηρεσίας. Υποθέστε ότι, βάσει της ανάλυσης και της σχεδίασης που έχουμε ήδη ολοκληρώσει, έχουμε καταλήξει πως η εφαρμογή μας θα εκτελείται σε προσωπικούς υπολογιστές και, όσον αφορά τους τελικούς χρήστες, θα χωρίζεται σε δύο υποσυστήματα: το υποσύστημα επικοινωνίας και το υποσύστημα διεπαφής (interface). Σχεδιάστε ένα διάγραμμα διάταξης για την εφαρμογή.

## ΕΝΟΤΗΤΑ 9.3. ΒΗΜΑΤΑ ΣΤΗ ΣΧΕΔΙΑΣΗ

Έχοντας παρουσιάσει τις έννοιες και τα εκφραστικά εργαλεία που μας παρέχει η ενοποιημένη προσέγγιση ανάπτυξης λογισμικού για τη διαδικασία της σχεδίασης, μπορούμε να περάσουμε στα βήματα που μας επιτρέπουν να μεταβούμε από την ανάλυση στη σχεδίαση. Στη διαδικασία αυτή, δεδομένα εισόδου είναι το μοντέλο ανάλυσης και τα στοιχεία που το αποτελούν, όπως αυτά έχουν παρουσιαστεί στο Κεφάλαιο 3, ενώ κατά την έξοδο της διαδικασίας λαμβάνουμε το μοντέλο σχεδίασης, όπως αυτό περιγράφεται στο παρόν κεφάλαιο. Έχοντας στα χέρια μας το μοντέλο σχεδίασης, μπορούμε να προχωρήσουμε στην υλοποίηση, η οποία με τη σειρά της θα μας δώσει μια πρώτη λειτουργική έκδοση του συστήματος προς ανάπτυξη.

Ακολουθώντας αυτά τα βήματα και ενώ προσπαθούμε να δημιουργήσουμε το μοντέλο της σχεδίασης, οφείλουμε να κρατάμε κατά νου δύο δεδομένα που ισχύουν και στις άλλες μεταβατικές φάσεις από μία διαδικασία σε κάποια άλλη:

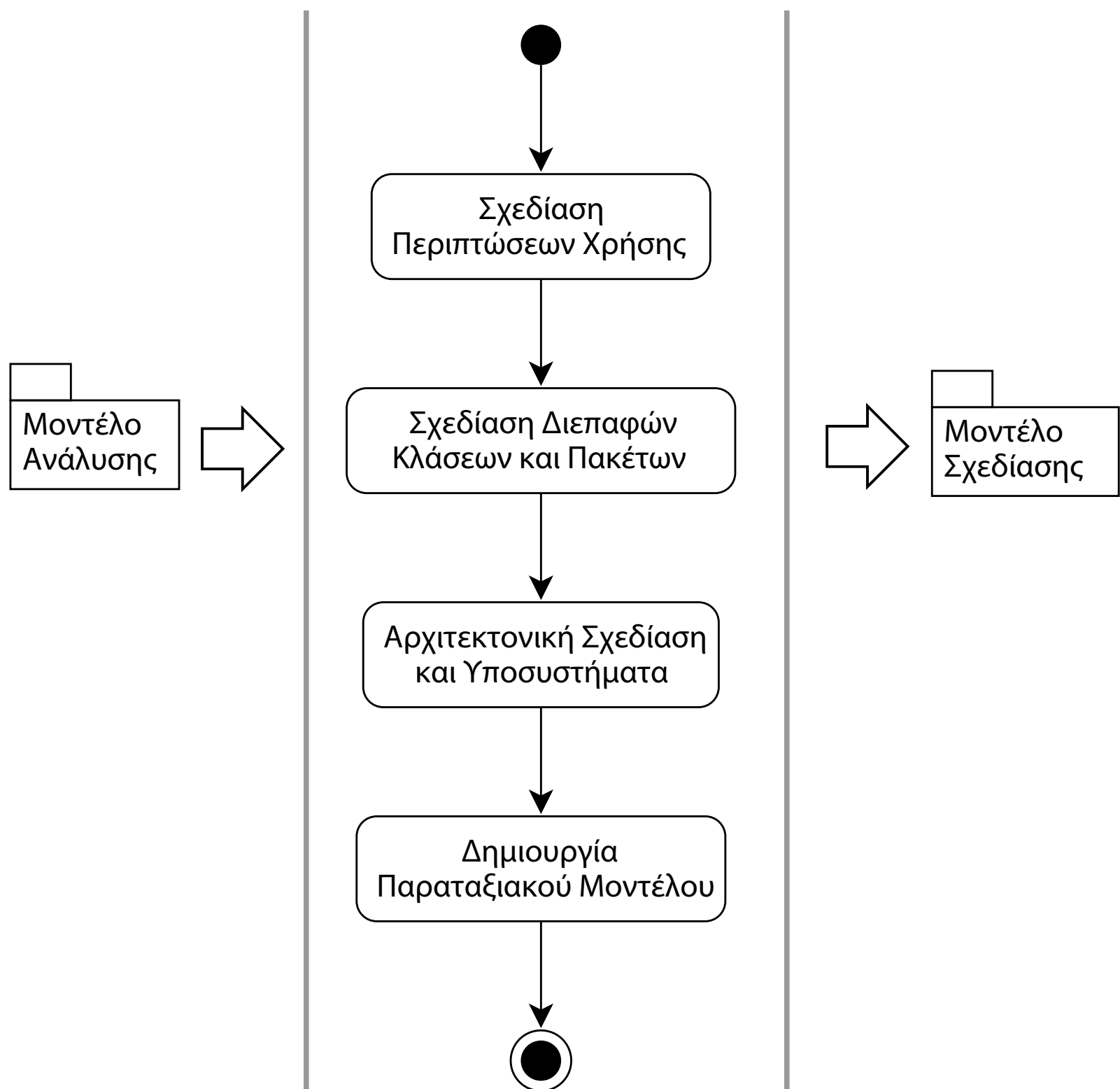
- 1) Δεν υπάρχει μία μοναδική ορθή μετάβαση ούτε ένα μοναδικό ορθό μοντέλο σχεδίασης δεδομένων ενός προβλήματος προς επίλυση και ενός μοντέλου ανάλυσης. Η ακριβής μεθοδολογία και τα βήματα που θα ακολουθηθούν θα εξαρτηθούν τόσο από την εφαρμογή που αναπτύσσεται όσο και από την ιδιοσυγκρασία του σχεδιαστή (μεταξύ πολλών άλλων παραμέτρων). Εδώ παρουσιάζεται μια μεθοδολογία που πιστεύουμε πως μπορεί να αποβεί χρήσιμη και διδακτική, χωρίς να είναι απαραίτητα και η βέλτιστη.
- 2) Όπως έχουμε αναφέρει σε διάφορα σημεία του βιβλίου, η ανάπτυξη λογισμικού είναι μια διαδικασία επαναληπτική και επαυξητική. Έτσι, οι διάφορες διαδικασίες, όπως αυτή της ανάλυσης, της σχεδίασης κ.λπ., δεν είναι πρακτικό να αντιμετωπιστούν απομονωμένα. Όπως έχουμε ήδη αναφέρει, το σενάριο κατά το οποίο ολοκληρώνουμε την ανάλυση και τη χρησιμοποιούμε αυτούσια σε όλα τα μεταγενέστερα στάδια της ανάπτυξης πρακτικά δεν υπάρχει. Σε μια ρεαλιστική κατάσταση, η σχεδίαση θα εγείρει ζητήματα τα οποία, προκειμένου να επιλυθούν, θα πρέπει να επαναπροσδιοριστούν και σημεία της ανάλυσης. Το ίδιο συμβαίνει σε όλα τα στάδια της



ανάπτυξης, αφού αποτελεί μια συνεχόμενη και επίπονη προσπάθεια και αλλαγές ή προβλήματα σε κάποιο στάδιο είναι φυσικό να επηρεάσουν προηγούμενα στάδια κ.ο.κ.

Κατά την αντικειμενοστρεφή φιλοσοφία, και όπως έχουμε δει στο τρέχον κεφάλαιο, αναγνωρίζονται τέσσερα διακριτά βήματα στη σχεδίαση, όπως φαίνονται στο Σχήμα 9.4.

**Σχήμα 9.4** Βήματα στη σχεδίαση κατά την ενοποιημένη προσέγγιση.



Κατά τη σχεδίαση περιπτώσεων χρήσης, τα πακέτα ανάλυσης της κάθε περίπτωσης χρήσης μεταφέρονται στο μοντέλο σχεδίασης. Πρακτικά, αυτό σημαίνει πως το κάθε πακέτο ελέγχεται πως πληροί τις προϋποθέσεις της μελλοντικής υλοποίησης και συγκεκριμενοποιείται περαιτέρω. Αυτός ο έλεγχος και η συγκεκριμενοποίηση συμπεριλαμβάνει και τις υπάρχουσες κλάσεις και διεπαφές. Άρα, η σχεδίαση κλάσεων και διεπαφών ουσιαστικά ανήκει στη σχεδίαση περιπτώσεων χρήσης. Εδώ επιλέγουμε να παρουσιάσουμε τα βήματα αυτά ως διακριτά εξαιτίας της σημαντικότητάς τους. Η σχεδίαση περιπτώσεων χρήσης μάς αφήνει ένα σημείο αναφοράς με την αρχική ανάλυση του προβλήματος, ενώ η σχεδίαση των διεπαφών, των κλάσεων και τελικά των πακέτων μάς φέρνει πιο κοντά στην υλοποίηση. Για τον σκοπό αυτής της εισαγωγής θα κάνουμε την παραδοχή ότι το μοντέλο περιπτώσεων χρήσης παραμένει αμετάβλητο κατά τη σχεδίαση και θα συνεχίσουμε στη σχεδίαση των αντίστοιχων πακέτων. Αυτή η παραδοχή είναι ρεαλιστική, αφού αν κατά τη διάρκεια της σχεδίασης χρειαστεί να προβούμε στην αλλαγή της προδιαγραφής κάποιου πακέτου που αντιστοιχεί σε κάποια περίπτωση χρήσης, τότε, κατά πάσα πιθανότητα, οφείλουμε να διακόψουμε τη σχεδίαση του εν λόγω πακέτου και να επανεξετάσουμε την περίπτωση χρήσης από το επίπεδο της ανάλυσης, δηλαδή αναδρομικά. Βάσει των προηγούμενων βημάτων, η αρχιτεκτονική σχεδίαση μας βοηθά να διαχωρίσουμε τις διάφορες λογικές οντότητες της σχεδίασης σε υποσυστήματα με καλά ορισμένες λειτουργίες, που συνολικά θα μας δώσουν το θεμιτό αποτέλεσμα. Τέλος, η δημιουργία του μοντέλου εγκατάστασης θα μας δώσει μια πρακτική απεικόνιση της διάταξης των –μελλοντικά– υλοποιημένων τμημάτων λογισμικού που θα κληθούμε να εφαρμόσουμε στον εξοπλισμό που προβλέπεται από το έργο. Αυτή η σειρά βημάτων θα μας αφήσει σε μια θέση όπου η σταδιακή υλοποίηση του συστήματος να μπορεί να γίνει οργανωμένα και ελεγχόμενα.

### **9.3.1. Σχεδίαση περιπτώσεων χρήσης**

Ήδη από το στάδιο της ανάλυσης οι περιπτώσεις χρήσεις έχουν αναλυθεί σε πακέτα αποτελούμενα από κλάσεις. Το καθένα από αυτά τα πακέτα αντιστοιχεί συνήθως σε μία περίπτωση χρήσης. Επιπλέον, οι κλάσεις της

ανάλυσης έχουν κατηγοριοποιηθεί σε κλάσεις οντοτήτων, ελέγχου και συνοριακές, και οι περιπτώσεις χρήσεις έχουν περιγραφεί βάσει διαγραμμάτων συνεργασίας. Αυτές οι κλάσεις όμως βρίσκονται ακόμα σε αρχικό επίπεδο και η περιγραφή τους είναι αρκετά αφηρημένη. Είναι προφανές πως η ανάλυση των κλάσεων βρίσκεται προσκολλημένη στο πεδίο του προβλήματος και όχι στη λύση που προσπαθούμε να επιτύχουμε. Ο σκοπός της σχεδίασης είναι η μετάβαση από το πεδίο του προβλήματος στο πεδίο της λύσης. Έτσι, με βάση τις κλάσεις της ανάλυσης θα προχωρήσουμε στις κλάσεις της σχεδίασης, ούτως ώστε η κάθε περίπτωση χρήσης να προδιαγραφεί με τέτοιο τρόπο που να καταστεί υλοποιήσιμη. Προφανώς, υπάρχουν διάφοροι τρόποι να μεταφέρουμε μία περίπτωση χρήσης στη σχεδίαση, και η διαδικασία είναι, όπως έχουμε τονίσει πολλές φορές, επαυξητική και επαναληπτική. Έχοντας ως εργαλεία κλάσεις και διεπαφές, μια αρχική προσέγγιση θα μπορούσε να είναι η ακόλουθη:

- i. Αρχικά διατηρούμε τις περιπτώσεις χρήσης και τα πακέτα τους ως έχουν.
- ii. Για κάθε περίπτωση χρήσης αναλογιζόμαστε τις κλάσεις ανάλυσης που έχουμε δημιουργήσει. Αντιμετωπίζουμε τους διαφορετικούς τύπους κλάσεων ως εξής:
  - α. Οι συνοριακές κλάσεις θα μετατραπούν σε κλάσεις σχεδίασης, οι οποίες όμως θα διέπονται από διεπαφές. Αναλόγως με τις ανάγκες της εφαρμογής, μπορούμε να δημιουργήσουμε πολλές κλάσεις που θα υλοποιούν την κάθε διεπαφή, όμως θα υπάρχει μία διεπαφή για καθεμία συνοριακή κλάση.
  - β. Οι κλάσεις ελέγχου θα μετατραπούν σε κλάσεις οι οποίες, αρχικά, δεν είναι απαραίτητο να έχουν εσωτερική κατάσταση, δηλαδή δεν θα είναι παρά συλλογές από μεθόδους οι οποίες θα έχουν, θεωρητικά τουλάχιστον, σημασιολογική συνοχή. Αυτό σημαίνει πως κάθε τέτοια κλάση θα μπορεί να δημιουργεί ένα μοναδικό αντικείμενο κατά την εκτέλεση της εφαρμογής. Αυτό μπορεί να αλλάξει μέχρι να κατασταλάξουμε στο τελικό σύνολο κλάσεων που θα χρειαζόμαστε και στη συμβολή τους στην εφαρμογή. Σε κάποιες μεθοδολογίες οι μέθοδοι των κλάσεων αυτών (και τελικά οι κλάσεις) μπορεί να ενσωματώνονται ως μέθοδοι σε άλλες κλάσεις.

- γ. Οι κλάσεις οντοτήτων θα μετατραπούν σε ολοκληρωμένες κλάσεις με εσωτερική κατάσταση και μεθόδους.
- iii. Με βάση την περίπτωση χρήσης που εξετάζουμε, «γεμίζουμε» την κάθε κλάση σχεδίασης με τα απολύτως απαραίτητα πεδία και μεθόδους που θα μπορέσουν να ικανοποιήσουν τις ανάγκες της περίπτωσης χρήσης. Για τον σκοπό αυτό είναι χρήσιμο να συμβουλευόμαστε και τα διαγράμματα συνεργασίας που έχουμε από την ανάλυση, καθώς και τα διαγράμματα ακολουθίας που δημιουργούμε κατά τη σχεδίαση.
- iv. Για κάθε περίπτωση χρήσης δημιουργούμε ένα διάγραμμα κλάσεων και ένα διάγραμμα σειράς (sequence diagram). Αυτό θα μας βοηθήσει να εξακριβώσουμε την ορθότητα των μεθόδων που έχουμε εισάγει στις κλάσεις και να κατανοήσουμε περισσότερο τις ανάγκες της περίπτωσης χρήσης με βάση τα αντικείμενα που απαιτεί για την υλοποίησή της. Όταν έχουμε δημιουργήσει τα διαγράμματα κλάσεων για όλες τις περιπτώσεις χρήσης, μπορούμε να τα ενώσουμε σε ένα για όλη την εφαρμογή ή, στην περίπτωση που η εφαρμογή είναι μεγάλη, μπορούμε να περιοριστούμε σε ένα διάγραμμα κλάσης ανά υποσύστημα.
- v. Εάν η περίπτωση χρήσης ικανοποιείται, μπορούμε να συνεχίσουμε με την επόμενη.
- Όταν έχουμε δημιουργήσει κλάσεις και διεπαφές σχεδίασης για όλες τις περιπτώσεις χρήσης, ελέγχουμε πως δεν έχουμε πανομοιότυπες κλάσεις ή διεπαφές σε πολλαπλά μέρη. Αν αυτό συμβαίνει, τότε πρέπει να τα συμπτύξουμε με τέτοιο τρόπο ώστε οι εμπλεκόμενες περιπτώσεις χρήσης να συνεχίσουν να ικανοποιούνται. Τέτοιες συμπτύξεις θα μας καθοδηγήσουν και αργότερα, όταν θα καταλήξουμε στα υποσυστήματα της εφαρμογής μας. Η λογική πίσω από αυτή την κίνηση είναι απλή: αν δύο πακέτα έχουν κοινές κλάσεις, τότε ίσως σε συνδυασμό καλύπτουν μια σειρά συναφών λειτουργιών και, άρα, ίσως να έπρεπε να ανήκουν στο ίδιο υποσύστημα.
- Εάν διαπιστώσουμε πως οι κλάσεις ή τα πακέτα μας δεν επαρκούν για να ικανοποιήσουν μια περίπτωση χρήσης ή πως έχουμε συχνά αλληλεπικαλυπτόμενες κλάσεις ή πακέτα που ικανοποιούν διαφορετικές περιπτώσεις

χρήσης, ίσως είναι χρήσιμο να επανεξετάσουμε τις περιπτώσεις χρήσης, να προβούμε σε αλλαγές και να επαναλάβουμε την ανάλυση.

## Μελέτη περίπτωσης

Θα εφαρμόσουμε την παραπάνω μεθοδολογία στο πρόγραμμα «Επίκουρος» για την περίπτωση χρήσης πεδίου ανάλυσης «διαγραφή σπουδαστή». Η ανάλυση αυτής της περίπτωσης χρήσης έχει δοθεί στην Ενότητα 8.5.2. Το αντίστοιχο διάγραμμα συνεργασίας φαίνεται στο Σχήμα 8.22 του προηγούμενου κεφαλαίου.

Από τις συνοριακές κλάσεις της μελέτης προκύπτουν οι διεπαφές «Δ-Εγγραφές», από την κλάση «Class Interface εγγραφής», και «Δ-Σπουδαστές», από την κλάση «Class Interface σπουδαστή». Οι υπόλοιπες συνοριακές κλάσεις που έχουμε αναγνωρίσει αποτελούν διεπαφικά εργαλεία για τον χρήστη (τον χειριστή στη γραμματεία) και δεν θα μας απασχολήσουν στην παρούσα μελέτη.

Από την κλάση ελέγχου «διαγραφή σπουδαστή» προκύπτει η κλάση σχεδίασης «έλεγχος σπουδαστών». Βλέποντας το διάγραμμα συνεργασίας, σκεφτόμαστε πως αυτή θα περιέχει μεθόδους που θα μας επιτρέψουν να διαγράψουμε σπουδαστές και που θα συντονίζουν διάφορες αναζητήσεις σε λίστες εγγραφών, σπουδαστών κ.λπ. Δεν περιορίζουμε το όνομά της στην πράξη της διαγραφής, αφού η γραμματεία θα χρειαστεί να διαχειρίζεται επιπλέον παρόμοιες εντολές και σκεφτόμαστε πως μια τέτοια κλάση μπορεί να μας φανεί χρήσιμη και για άλλες περιπτώσεις χρήσης.

Τέλος, από τις κλάσεις οντοτήτων προκύπτουν οι κλάσεις σχεδίασης «εγγραφές» και «σπουδαστές». Αυτές οι κλάσεις θα αντικατοπτρίζουν κάποιας μορφής λίστα δεδομένων, της οποίας η υλοποίηση δεν μας αφορά σε αυτή τη φάση. Όμως, όπως προκύπτει από το διάγραμμα συνεργασίας, θα πρέπει να υποστηρίζουν ενέργειες όπως εύρεση, διαγραφή κ.λπ. Με άλλα λόγια, θα πρέπει να υπόκεινται στις διεπαφές που έχουμε αντιστοιχίσει, τις «Δ-Εγγραφές» και «Δ-Σπουδαστές».

Έχοντας αναγνωρίσει τις κλάσεις που μας ενδιαφέρουν και έχοντάς τις μεταφέρει στο μοντέλο σχεδίασης, μπορούμε να περάσουμε στο επόμενο

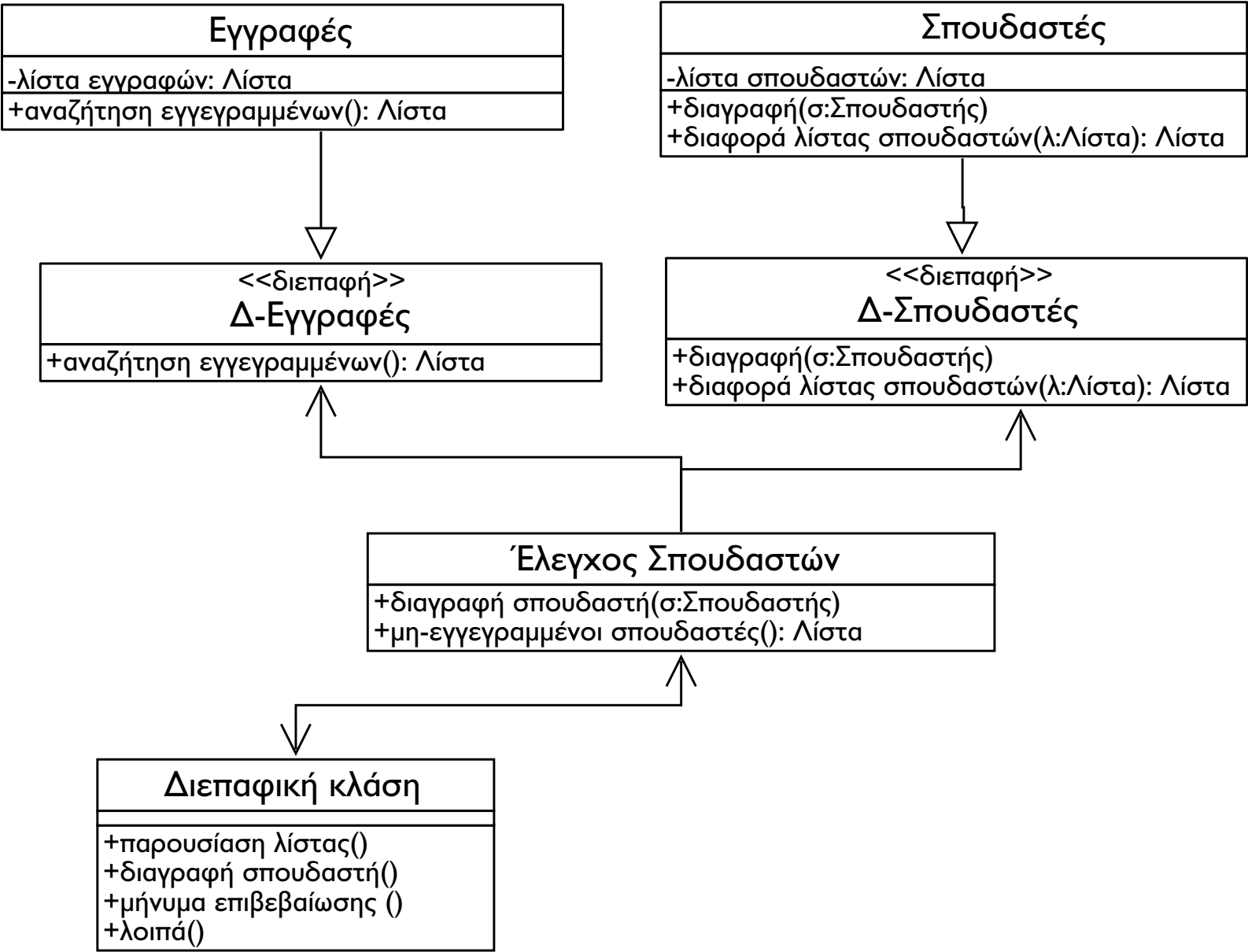
βήμα, όπου θα δώσουμε στις κλάσεις μεθόδους και, αν κρίνουμε σκόπιμο, πεδία. Ας ξεκινήσουμε με την ίδια σειρά, δηλαδή από τις διεπαφές. Για τη διεπαφή «Δ-Εγγραφές» αναγνωρίζουμε τη μέθοδο «αναζήτηση εγγεγραμμένων σπουδαστών», ενώ για τη διεπαφή «Δ-Σπουδαστές» αναγνωρίζουμε τις μεθόδους «διαγραφή» και «διαφορά λίστας σπουδαστών». Αυτές οι μέθοδοι θα πρέπει να υποστηρίζονται και από τις αντίστοιχες υλοποιήσιμες κλάσεις «εγγραφές» και «σπουδαστές», που τελικά θα μας εφοδιάσουν και με τα αντικείμενα που θα φέρουν εις πέρας το έργο.

Η κλάση ελέγχου «έλεγχος σπουδαστών» θα έχει τις μεθόδους «διαγραφή σπουδαστή», που θα διευθύνει την ενέργεια της διαγραφής, και τη μέθοδο «μη εγγεγραμμένοι σπουδαστές», που θα επιστρέφει μια λίστα σπουδαστών την οποία θα μπορεί το σύστημα να διαγράψει.

Έτσι, καταλήγουμε στο παρακάτω διάγραμμα κλάσεων:



**Σχήμα 9.5** Διάγραμμα κλάσεων που προέκυψε από τη σχεδίαση της περίπτωσης χρήσης «διαγραφή σπουδαστή» του προγράμματος «Επίκουρος».

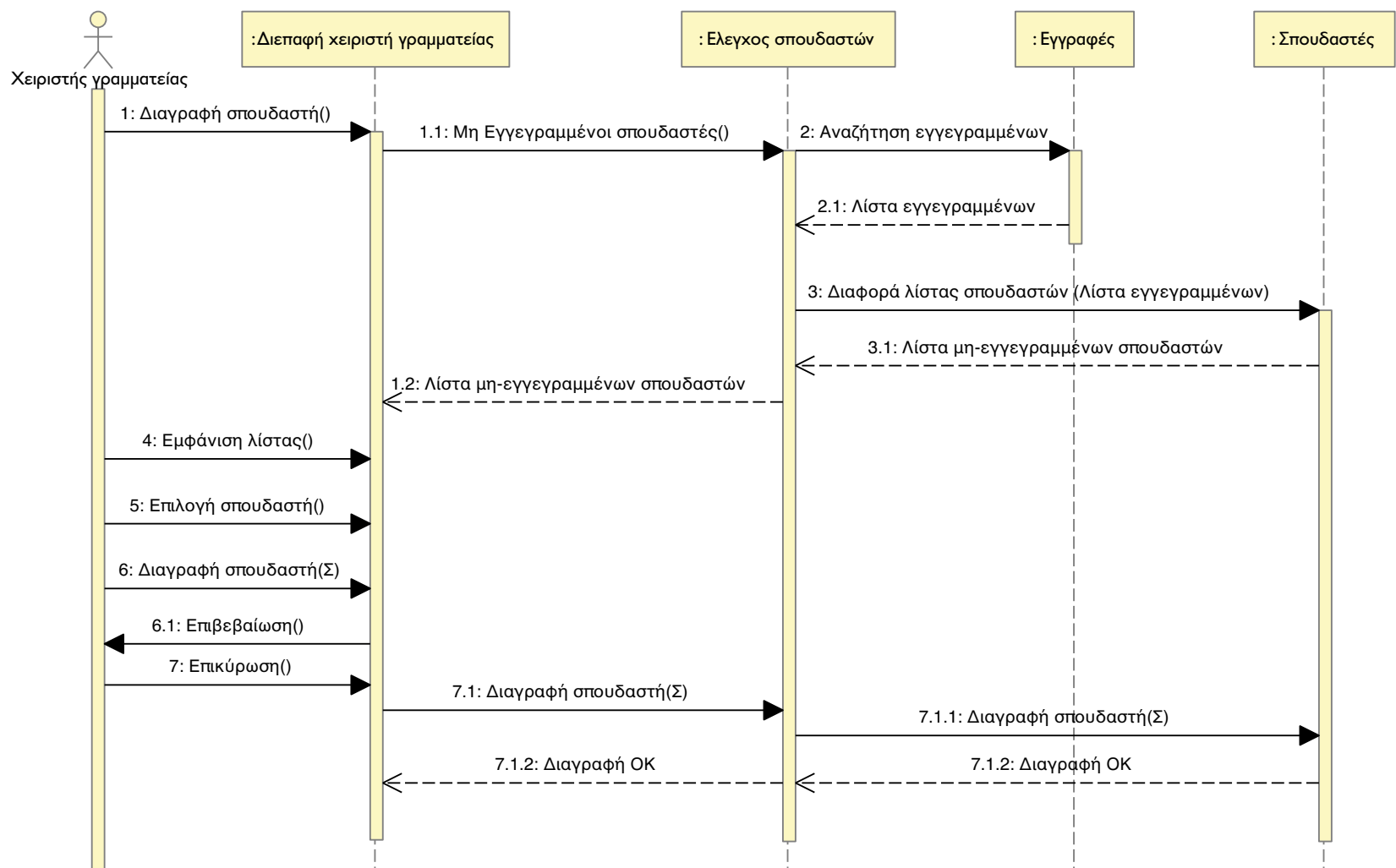




Σε αυτό το διάγραμμα, και για το σκοπό της διαφοροποίησης της περίπτωσης χρήσης από τη διεπαφή με τους τελικούς χρήστες, έχουμε εισάγει μία γενική κλάση «διεπαφική κλάση», η οποία έχει κάποιες μεθόδους για εισαγωγή και έξοδο δεδομένων. Αυτές οι μέθοδοι δεν έχουν προδιαγραφεί πλήρως, απλώς υπάρχουν για να μας βοηθήσουν να μοντελοποιήσουμε την περίπτωση χρήσης.

Έχοντας σχεδιάσει το διάγραμμα κλάσεων, θα περάσουμε στη δημιουργία ενός διαγράμματος ακολουθίας, σημειώνοντας ότι δεν είναι πάντα αυτή η σειρά με την οποία είναι απαραίτητο να κατασκευάζονται τα διαγράμματα αυτά. Ενδέχεται, κατά τη σχεδίαση της περίπτωσης χρήσης στην οποία βρισκόμαστε να έχουμε ήδη ένα διάγραμμα ακολουθίας το οποίο σε κάθε περίπτωση θα είναι συνεπές με το διάγραμμα κλάσεων, δηλαδή θα χρησιμοποιήσει τις μεθόδους που εισάγαμε στις κλάσεις. Το παρακάτω σχήμα απεικονίζει ένα διάγραμμα ακολουθίας που καλύπτει την περίπτωση που η διαγραφή του σπουδαστή γίνει επιτυχώς.

**Σχήμα 9.6** Διάγραμμα ακολουθίας που περιγράφει την περίπτωση χρήσης «διαγραφή σπουδαστή» του προγράμματος «Επίκουρος».



Σε αυτό το σημείο, η περίπτωση χρήσης ικανοποιείται, άρα μπορούμε να πούμε πως η σχεδίασή της ολοκληρώθηκε.

## **Δραστηριότητα 5/Κεφάλαιο 9**

Κατασκευάστε διαγράμματα κλάσεων και ακολουθίας για την περίπτωση χρήσης «διαγραφή μαθήματος» του λογισμικού «Επίκουρος».

## **Άσκηση 2/Κεφάλαιο 9**

Για την εφαρμογή «Επίκουρος» και την περίπτωση χρήσης «εκτύπωση βαθμολογίας μαθήματος», παράγετε τα σχετικά διαγράμματα κλάσεων και ακολουθίας. Στη λύση σας υποθέστε πως η προηγούμενη ανάλυση της συγκεκριμένης περίπτωσης χρήσης μάς παρέχει την παρακάτω ροή:

1. Ο χειριστής «χειριστής γραμματείας» επιλέγει από το μενού την εντολή «προβολή μαθημάτων».
2. Ο χειριστής «χειριστής γραμματείας» επιλέγει ένα από τα μαθήματα.
3. Ο χειριστής «χειριστής γραμματείας» επιλέγει από το μενού την εντολή «εκτύπωση βαθμολογίας».
4. Ο «Επίκουρος» συλλέγει και εκτυπώνει όλους τους εγγεγραμμένους σπουδαστές και τις βαθμολογίες τους για το επιλεγμένο μάθημα.
5. Ο «Επίκουρος» ενημερώνει τον χειριστή πως η εκτύπωση ολοκληρώθηκε.

### 9.3.2. Αρχιτεκτονική σχεδίαση και υποσυστήματα

Η αρχιτεκτονική σχεδίαση και ο διαχωρισμός του συστήματος σε υποσυστήματα μπορεί να γίνει πριν ή μετά τη σχεδίαση των περιπτώσεων χρήσης. Όπως έχει προαναφερθεί, η απόφαση αυτή ανήκει κυρίως στον σχεδιαστή και εξαρτάται από διάφορους παράγοντες. Εδώ προτιμούμε να προβούμε στην αρχιτεκτονική σχεδίαση μετά, γιατί κατά τη σχεδίαση των περιπτώσεων χρήσης υπάρχει περίπτωση να διορθώσουμε και να προσθέσουμε κλάσεις στο μοντέλο σχεδίασης. Πολλές φορές αυτό έχει ως αποτέλεσμα την αλλαγή και των πακέτων που έχουμε, τα οποία θα μας υπαγορεύσουν σε μεγάλο βαθμό και τα υποσυστήματα που θα χρειαστούμε. Όπως και στη σχεδίαση των περιπτώσεων χρήσης, έτσι και στην αρχιτεκτονική σχεδίαση, δεν υπάρχει μία μοναδική ορθή λύση.

Η μεθοδολογία που θα ακολουθήσουμε για την αναγνώριση των υποσυστημάτων είναι απλή. Θα ξεκινήσουμε θεωρώντας ως υποσυστήματα τα πακέτα της σχεδίασης και θα προχωρήσουμε περαιτέρω με γνώμονα την εφαρμογή που αναπτύσσουμε και τα ειδικά χαρακτηριστικά της. Πολλές φορές τέτοια χαρακτηριστικά προέρχονται και από τις μη λειτουργικές ανάγκες. Για παράδειγμα, αν η εφαρμογή μας βασίζεται σε μία εξωτερική βάση δεδομένων, ίσως πρέπει να δημιουργήσουμε ένα υποσύστημα που θα ελέγχει και θα χρησιμοποιεί αυτή τη βάση δεδομένων. Μια άλλη προσέγγιση βασίζεται στους χρήστες της εφαρμογής και τις διαφορετικές όψεις που έχουν. Για παράδειγμα, σε μία διαδικτυακή πύλη που υποστηρίζει διαφορετικά επίπεδα χρηστών, όπως ελεγκτών, μελών, επισκεπτών κ.λπ., μπορεί να αναγνωρίσουμε υποσυστήματα που παρέχουν λειτουργίες στις διαφορετικές ομάδες χρηστών κ.ο.κ.

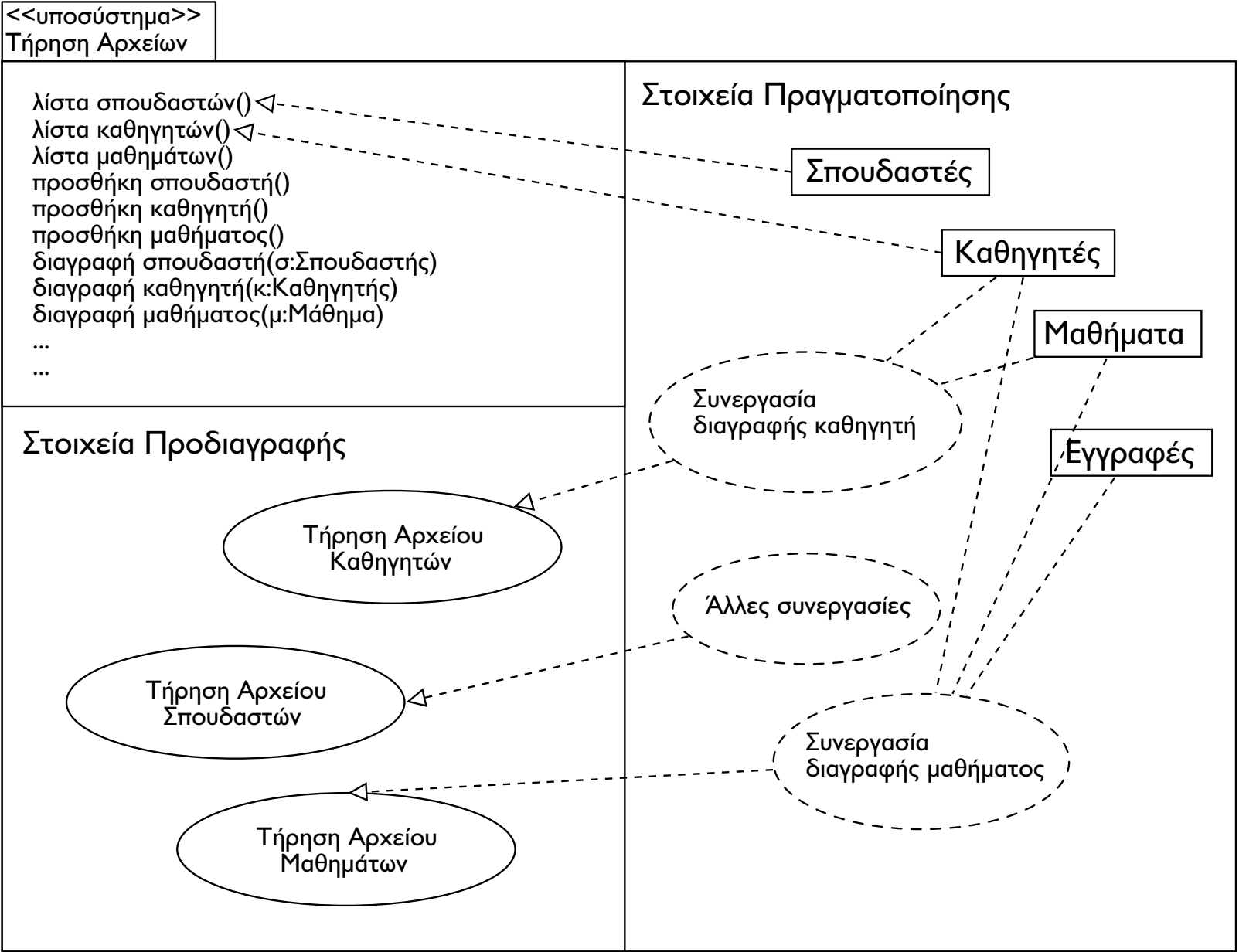
Σε κάθε περίπτωση, τα υποσυστήματα που αναγνωρίζουμε στη σχεδίαση πρέπει να παρέχουν κάτι συγκεκριμένο στο συνολικό σύστημα και οι λειτουργίες του από τα άλλα υποσυστήματα να είναι απολύτως διακριτές. Γενικά, κάθε υποσύστημα χρειάζεται να έχει τρία χαρακτηριστικά: την περιγραφή κάποιας διεπαφής για να επικοινωνεί με τα άλλα υποσυστήματα, έναν αριθμό κλάσεων που συντελούν στην εργασία που επιτελεί το υποσύστημα και έναν αριθμό περιπτώσεων χρήσης που ικανοποιεί. Επειδή η έννοια του

υποσυστήματος είναι συνυφασμένη με την έννοια του πακέτου κλάσεων, θα χρησιμοποιήσουμε τον ίδιο συμβολισμό με τα πακέτα, συγκεκριμενοποιώντας το όμως με τη χρήση του στερεότυπου «υποσύστημα».

### **Μελέτη περίπτωσης**

Επιστρέφοντας στην εφαρμογή «Επίκουρος» και, συγκεκριμένα, στη δομή της ανάλυσής της (Σχήμα 8.17), ας υποθέσουμε πως έχει προηγηθεί η σχεδίαση του πακέτου «τήρηση αρχείων» και των συμπεριλαμβανομένων περιπτώσεων χρήσης. Τώρα, ας υποθέσουμε πως στο πακέτο αυτό συμπεριλαμβάνονται οι κλάσεις «εγγραφές», «μαθήματα», «καθηγητές» και «σπουδαστές», όπως επίσης και οι συναφείς τους μέθοδοι για την τήρηση του αρχείου, όπως «προσθήκη», «διαγραφή» κ.λπ. Εάν δεχτούμε πως η τήρηση αρχείου μπορεί να αποτελέσει ένα υποσύστημα, τότε αυτό προδιαγράφεται όπως στο παρακάτω σχήμα.

**Σχήμα 9.7** Το υποσύστημα «τήρηση αρχείων» της εφαρμογής «Επίκουρος».



Το σχήμα αυτό είναι προφανώς ημιτελές, όμως η σημασία του είναι εμφανής. Το υποσύστημα χωρίζεται στη διεπαφή (είναι το τμήμα πάνω αριστερά) μέσω της οποίας θα επιτυγχάνεται η επικοινωνία με άλλα υποσυστήματα. Τα στοιχεία πραγματοποίησης περιέχουν τις κλάσεις που συνεργάζονται, ενώ τα στοιχεία προδιαγραφής είναι οι περιπτώσεις χρήσης που ικανοποιούνται. Στα στοιχεία πραγματοποίησης ανήκουν επίσης και τα διαγράμματα συνεργασίας μεταξύ των κλάσεων για τα οποία δίνονται παραπομπές μέσα στις διακεκομμένες ελλείψεις. Όπως είναι φανερό, ένα τέτοιο σχήμα γίνεται σύντομα δυσανάγνωστο. Για τον λόγο αυτό προτείνουμε τη χρήση απλού κειμένου ή ενός πίνακα για την περιγραφή του.

Είναι καλή πρακτική το κάθε στοιχείο της ανάλυσης και της σχεδίασης να φέρει και έναν μοναδικό κωδικό. Έτσι, οι παραπομπές στα διάφορα εμπλεκόμενα στοιχεία (όπως κλάσεις, διαγράμματα συνεργασίας κ.λπ.) γίνονται πιο εύκολα και σε μικρότερο χώρο. Αν θέλετε να πάτε λίγο παραπέρα, μπορείτε να δοκιμάσετε να ολοκληρώσετε το παραπάνω σχήμα του υποσυστήματος. Μια σκέψη είναι να χρησιμοποιήσετε κάποιο ευρετήριο κλάσεων, διαγραμμάτων συνεργασίας και περιπτώσεων χρήσης, το οποίο θα συνοδεύσει το σχήμα.

### 9.3.3. Δημιουργία μοντέλου διάταξης

Το τελευταίο βήμα της σχεδίασης είναι η αρχική δημιουργία ενός μοντέλου διάταξης ή εγκατάστασης (deployment) το οποίο θα απεικονίζει την «ανάθεση» των συστατικών στοιχείων του λογισμικού στα διάφορα μέρη του εξοπλισμού. Η διάταξη αυτή πιθανώς θα αλλάξει καθώς θα προχωρά η υλοποίηση, αλλά είναι σημαντικό να την έχουμε δημιουργήσει λίγο πριν ξεκινήσει η υλοποίηση, αφού θα αποτελέσει έναν χρήσιμο οδηγό για τους προγραμματιστές του συστήματος. Σε αυτήν τη φάση ακολουθούμε την απλή μέθοδο, σύμφωνα με την οποία το κάθε υποσύστημα αντιστοιχεί σε κάποιο συστατικό στοιχείο. Προσοχή πρέπει να δοθεί στην επικοινωνία των κόμβων λογισμικού, καθώς και στο είδος του κάθε υποσυστήματος.

## Μελέτη περίπτωσης

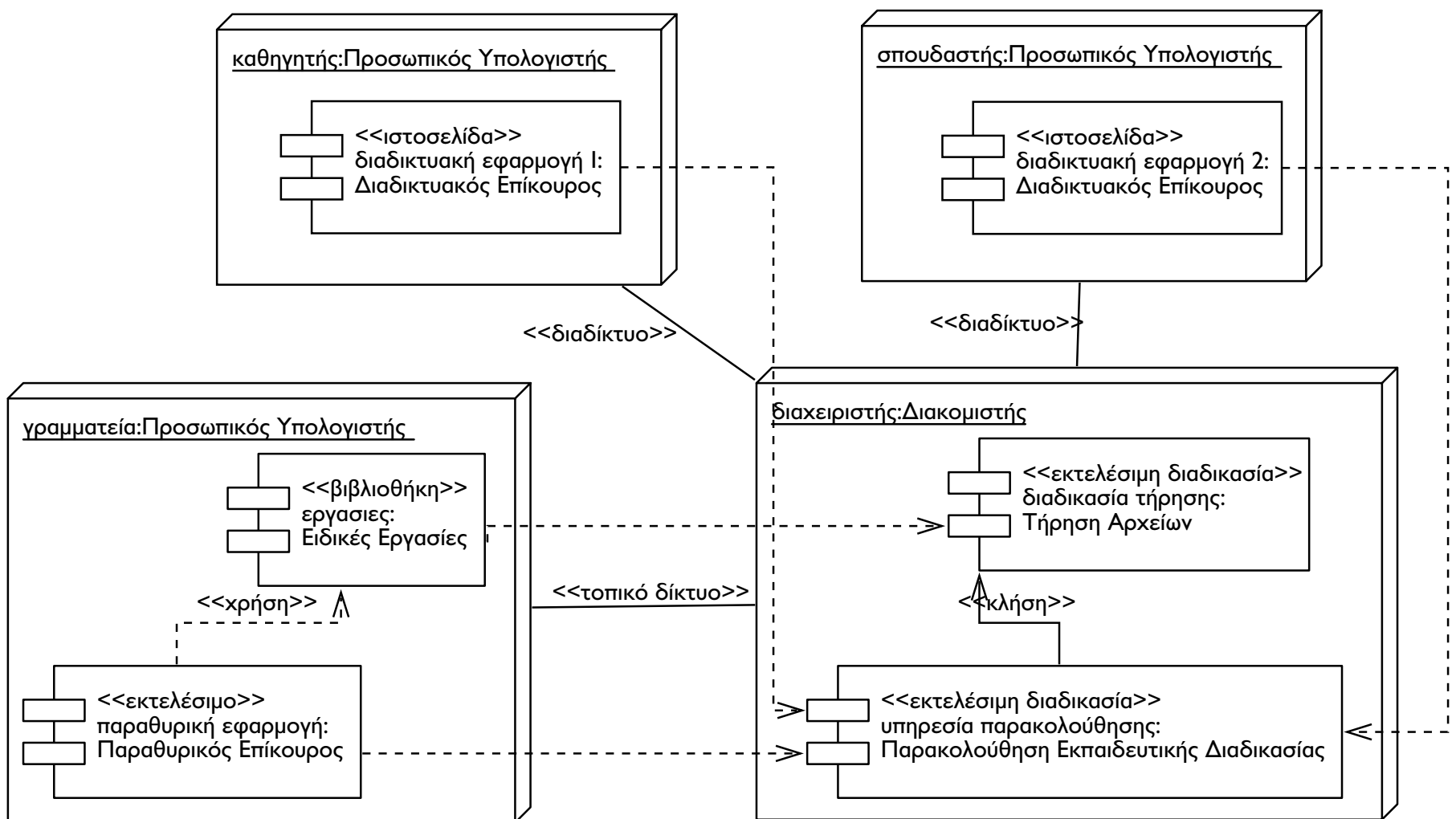
Θα επιστρέψουμε στην εφαρμογή «Επίκουρος» και θα χρησιμοποιήσουμε τα υπάρχοντα πακέτα που έχουν αναγνωριστεί κατά την ανάλυση ως συστατικά στοιχεία της εφαρμογής. Θα κάνουμε επίσης τις εξής παραδοχές:

1. Η τήρηση αρχείων θα γίνεται απομακρυσμένα, δηλαδή σε κάποιον διακομιστή και όχι τοπικά στον υπολογιστή του χρήστη της γραμματείας.
2. Η παρακολούθηση της εκπαιδευτικής διαδικασίας θα γίνεται και από καθηγητές και από σπουδαστές μέσω μιας δικτυακής πύλης. Αυτό σημαίνει πως θα υπάρχουν δύο καινούρια υποσυστήματα που θα παρέχουν διεπαφικές δυνατότητες στους χρήστες: ένα μέσω συμβατικού παραθυρικού περιβάλλοντος και ένα μέσω διαδικτύου.
3. Οι ειδικές εργασίες θα γίνονται μόνο από τη γραμματεία.

Αφού έχουμε κάνει αυτές τις παραδοχές, το διάγραμμα διάταξης της εφαρμογής μας θα μπορούσε να είναι το παρακάτω:



**Σχήμα 9.8** Το διάγραμμα διάταξης για την εφαρμογή «Επίκουρος».



### Άσκηση 3/Κεφάλαιο 9

Αν προσθέσουμε ένα ακόμα κανάλι χρήσης της εφαρμογής «Επίκουρος», για παράδειγμα, από κινητό τηλέφωνο, μπορείτε να περιγράψετε πώς θα άλλαζε το παραπάνω σχήμα; Αν το συστατικό στοιχείο «παρακολούθηση εκπαιδευτικής διαδικασίας» βρισκόταν στους επιμέρους υπολογιστές, πώς θα επηρεαζόταν η διαδικασία της εγκατάστασης;

## ΕΝΟΤΗΤΑ 9.4. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ

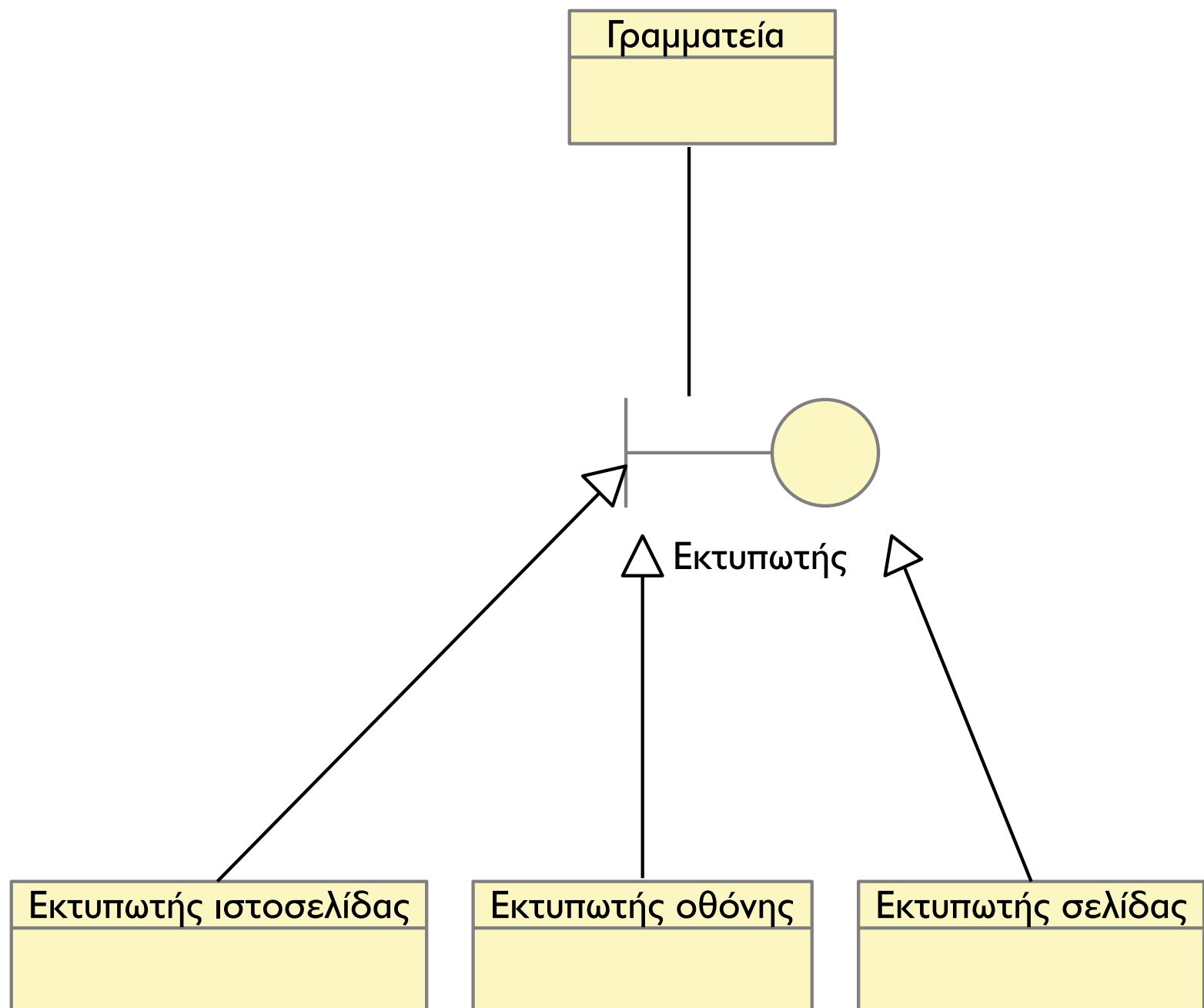
### Δραστηριότητα I/Κεφάλαιο 9

Από την περιγραφή της εφαρμογής προκύπτει πως η γραμματεία θα χρειάζεται να εκτυπώνει βαθμολογίες σπουδαστών, όπως επίσης και να τις παρουσιάζει στην οθόνη. Σε κάποια ενδεχόμενη επέκταση θα μπορούσαν να απαιτηθούν και επιπλέον εναλλακτικά μέσα για την εκτύπωση βαθμολογιών, όπως εμφάνιση σε ιστοσελίδες κ.ά. Επομένως, αν η εκτύπωση της βαθμολογίας (ή και άλλων στοιχείων) ενός σπουδαστή γίνεται αποκλειστικά μέσα από την εφαρμογή της γραμματείας, θα δυσκολέψουμε την επεκτασιμότητα και θα περιορίσουμε την ευελιξία της εφαρμογής. Αυτό γιατί, ενώ η εφαρμογή της γραμματείας απλώς χρειάζεται να στείλει κάποια βαθμολογία για εκτύπωση, θα πρέπει και να διαχειρίζεται τους ακριβείς τρόπους των διαφόρων εκτυπώσεων. Επιπλέον, η παραμικρή αλλαγή στον τρόπο εκτύπωσης βαθμολογιών θα επισύρει και μια σειρά αλλαγών στην εφαρμογή της γραμματείας, που θα είναι ήδη επιφορτισμένη με μια σειρά επιπρόσθετων λειτουργιών.

Η χρήση διεπαφών μάς επιτρέπει να προσεγγίσουμε αυτό το πρόβλημα με έναν πιο αποτελεσματικό τρόπο, ο οποίος μάλιστα είναι πιο κοντά στον φυσικό κόσμο του προβλήματος. Αν χρησιμοποιήσουμε τη διεπαφή «εκτυπωτής» να ορίζει μια μέθοδο «εκτύπωση σπουδαστή (σ: Σπουδαστής)», θα είμαστε σε θέση να έχουμε την εφαρμογή της γραμματείας να καλεί οποιονδήποτε εκτυπωτή χρειάζεται μέσω αυτής της διεπαφής. Έπειτα, θα

μπορούμε να δημιουργήσουμε διάφορες κλάσεις που θα υλοποιούν διαφορετικού τύπου εκτυπωτές μέσω της προκείμενης διεπαφής. Τέτοιες κλάσεις, για παράδειγμα, θα μπορούσαν να είναι οι εξής: «εκτυπωτής οθόνης», «εκτυπωτής χαρτιού», «εκτυπωτής ιστοσελίδας» κ.ο.κ. Η σχεδίαση αυτού του εναλλακτικού τρόπου εκτύπωσης φαίνεται στο Σχήμα 9.9. Με αυτόν τον τρόπο θα επιτυχάναμε μεγαλύτερη ευελιξία και επεκτασιμότητα.

**Σχήμα 9.9** Η εναλλακτική σχεδίαση της εκτύπωσης βαθμολογίας σπουδαστών για την εφαρμογή «Επίκουρος».

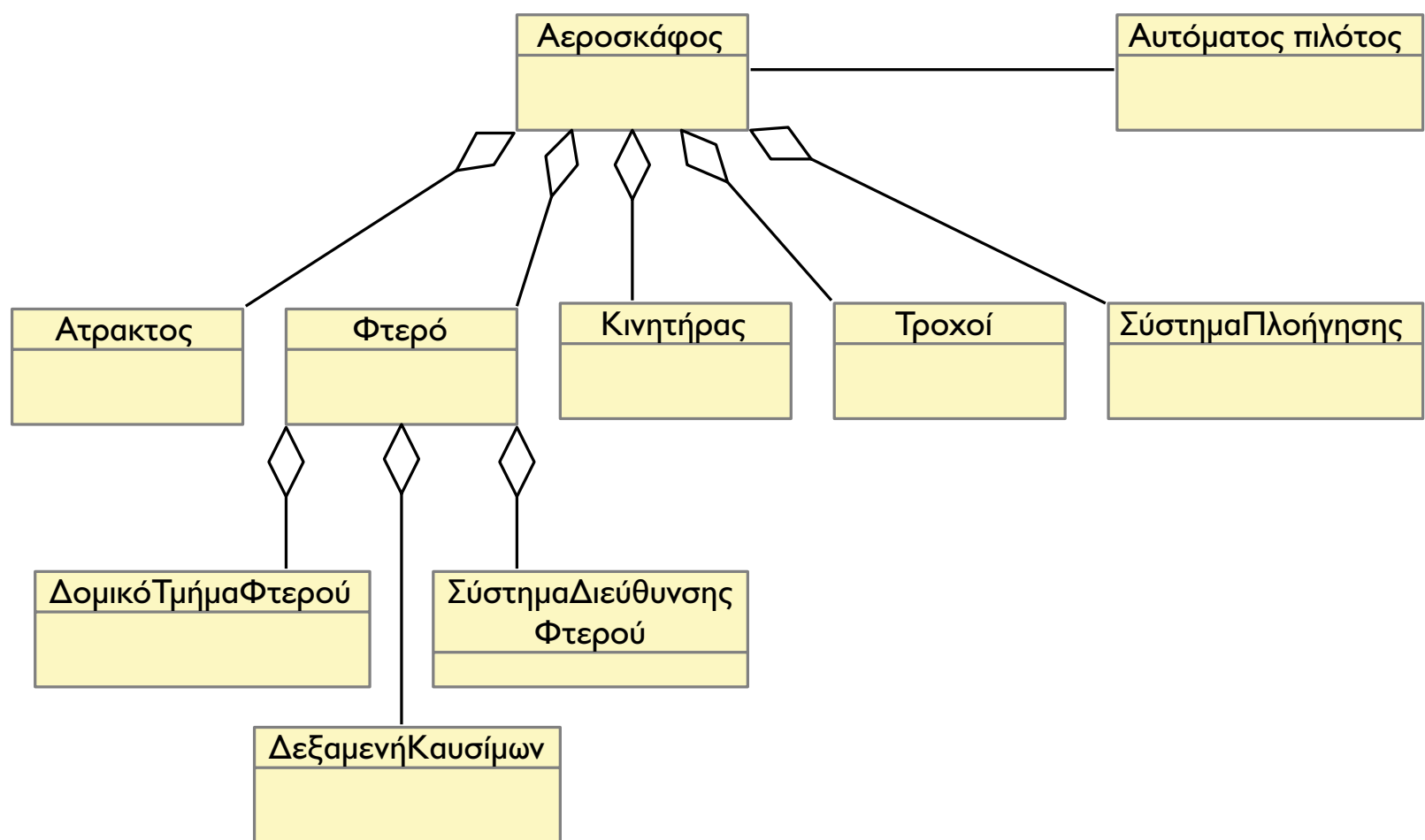


## Δραστηριότητα 2/Κεφάλαιο 9

---

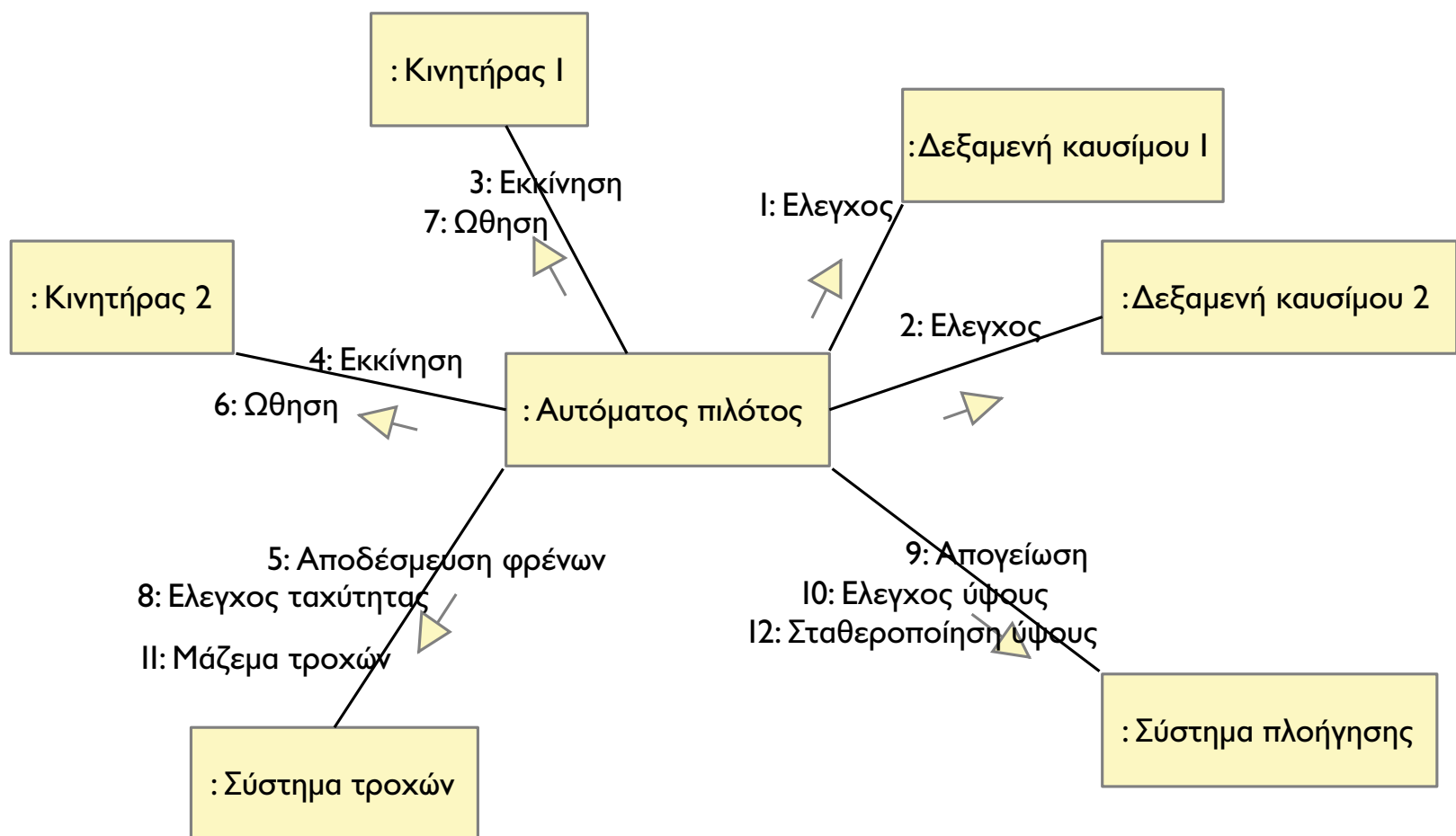
Το συγκεκριμένο διάγραμμα κλάσεων είναι ένα σωστό διάγραμμα μιας και μεταφέρει με άμεσο τρόπο τη φυσική σχέση συναρμολόγησης των διαφόρων εξαρτημάτων ενός αεροσκάφους στην αντικειμενοστρεφή λογική. Το διάγραμμα κλάσεων του παραδείγματος θα μπορούσε να χρησιμοποιηθεί αυτούσιο. Αποτελώντας μία καινούρια κλάση, ο αυτόματος πιλότος θα μπορούσε να επικοινωνεί με ένα αντικείμενο της κλάσης «αεροσκάφος», έτσι ώστε να χειριστεί τα διάφορα εξαρτήματα του αεροσκάφους ως εξής:

**Σχήμα 9.10** Παράδειγμα της σχέσης συναρμολόγησης.



Μοντελοποιώντας μια υποθετική περίπτωση χρήσης που θα επιτρέψει στον αυτόματο πιλότο να απογειώσει ο αεροσκάφος, είναι εύκολο να καταλήξουμε στο παρακάτω διάγραμμα συνεργασίας:

**Σχήμα 9.11** Ένα διάγραμμα συνεργασίας.



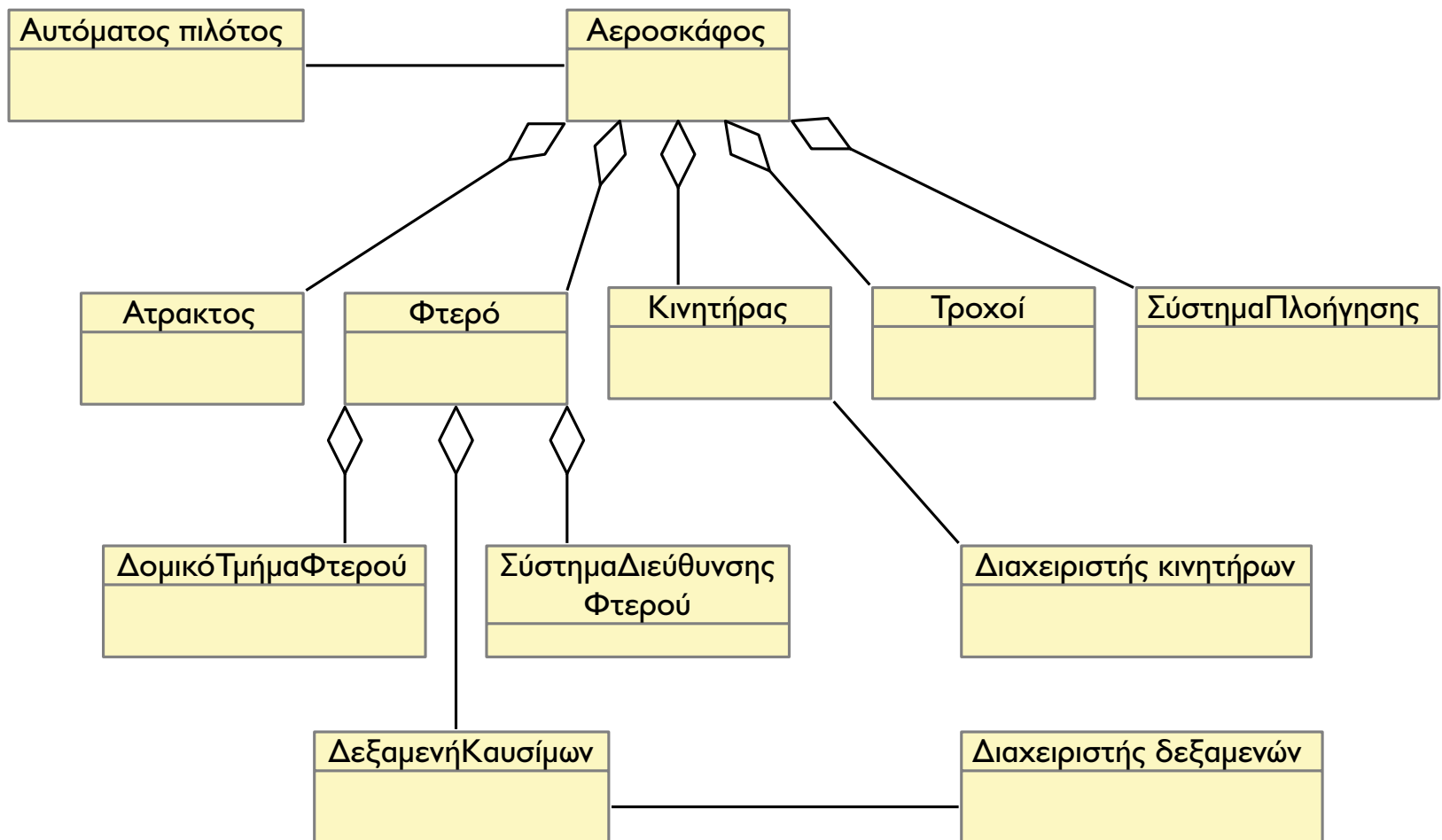


Έχοντας αυτό το διάγραμμα συνεργασίας, βλέπουμε πως το παραπάνω διάγραμμα κλάσεων θα μπορούσε, δεδομένων των απαραίτητων μεθόδων, να χρησιμοποιηθεί για την υλοποίηση της περίπτωσης χρήσης της απογείωσης. Όμως, παρατηρούμε πως επιδέχεται βελτιώσεων σε τουλάχιστον δύο σημεία:

1. Δεν θα ήταν λογικό για τον αυτόματο πιλότο να εκκινεί τους δύο (ή σε άλλο αεροσκάφος παραπάνω) κινητήρες ξεχωριστά για τον σκοπό της απογείωσης. Άρα, θα μπορούσε να βρεθεί ένας τρόπος, έτσι ώστε να αποφευχθεί αυτή η επανάληψη, αν είναι δυνατό. Το ίδιο θα έπρεπε να συμβεί και για τις δεξαμενές καυσίμου.
2. Από τη μεριά του διαγράμματος κλάσεων βλέπουμε πως η δεξαμενή καυσίμου ανήκει στην κλάση «φτερό». Όμως, όσον αφορά την περίπτωση χρήσης μας, αυτή η σχέση δεν είναι χρήσιμη. Με άλλα λόγια, ενώ αποτυπώνει τη δομή του αεροσκάφους, δεν αποτυπώνει τις ανάγκες μας για την επίλυση του προβλήματος.

Με αυτή την επιπλέον γνώση, που προέκυψε από την τριβή μας με το πρόβλημα, μπορούμε να καταλήξουμε στο εξής εναλλακτικό διάγραμμα κλάσεων:

**Σχήμα 9.12** Το τελικό διάγραμμα κλάσεων.



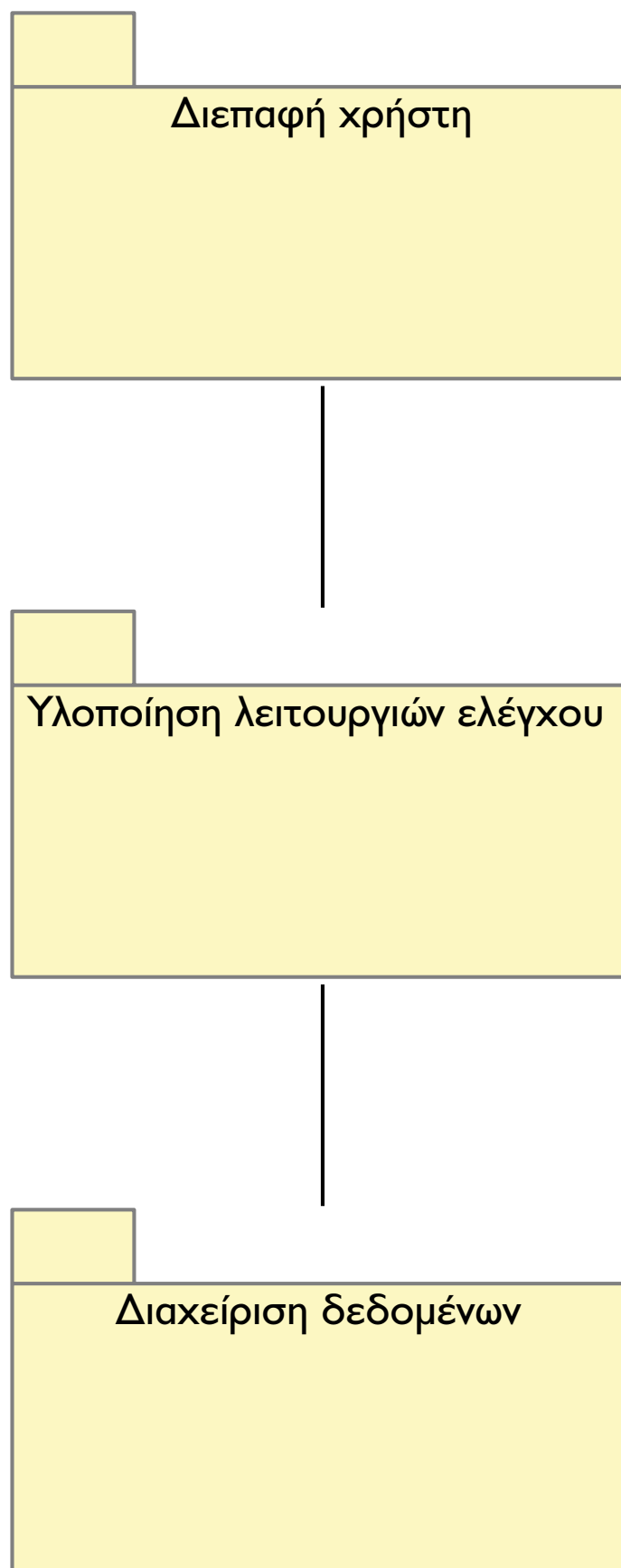
## Δραστηριότητα 3/Κεφάλαιο 9

---

Βάσει της περιγραφής που φαίνεται στο σχήμα 8.11, μπορούμε να χωρίσουμε το τελικό σύστημα σε τρία υποσυστήματα:

1. Υποσύστημα γραφικής διεπαφής χρηστών (Graphical User Interface – GUI).
2. Υποσύστημα ελέγχου και διεκπεραίωσης των διαδικασιών.
3. Βάση δεδομένων σπουδαστών και καθηγητών.

**Σχήμα 9.13** Ορισμός υποσυστημάτων.



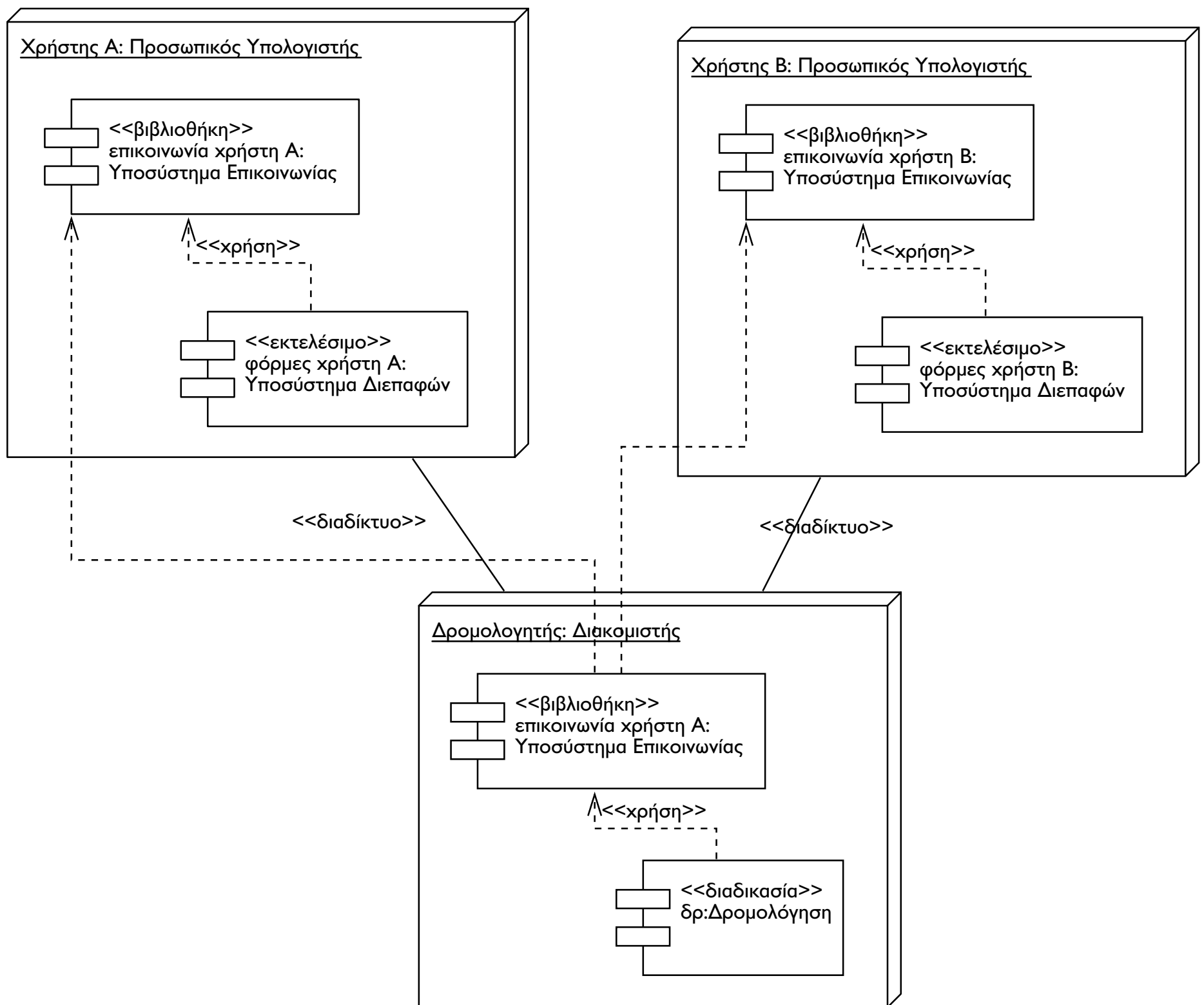
Το υποσύστημα GUI θα είναι υπεύθυνο για την παρουσίαση της εφαρμογής στους χρήστες/χειριστές της, δηλαδή στον χειριστή της γραμματείας. Θα παρέχει ένα παραθυρικό περιβάλλον μέσα από το οποίο ο χρήστης θα μπορεί να διεκπεραιώσει ό,τι προβλέπεται από τις σχετικές περιπτώσεις χρήσης.

## Δραστηριότητα 4/Κεφάλαιο 9

---

Το υποσύστημα επικοινωνίας θα αναλαμβάνει τη μετάδοση και λήψη δεδομένων φωνής, ενώ το υποσύστημα διεπαφής θα παρέχει στον χρήστη ένα περιβάλλον λειτουργίας της εφαρμογής. Στην αρχιτεκτονική μας, τα δεδομένα φωνής θα ταξιδεύουν μεταξύ των ομιλητών μέσω ενός κεντρικού διακομιστή που θα αναλαμβάνει να τα προωθεί στους σωστούς τελικούς χρήστες βάσει της υπάρχουσας σύνδεσης. Το μοντέλο διάταξης ενός τέτοιου σεναρίου φαίνεται στο επόμενο σχήμα.:

**Σχήμα 9.14** . Το μοντέλο διάταξης του προβλήματός μας.



Για τη διεκπεραίωση μιας υποτιθέμενης συνομιλίας μεταξύ δύο χρηστών Α και Β συμβάλλουν τρεις κόμβοι: οι δύο προσωπικοί υπολογιστές των χρηστών και ο διακομιστής που αναλαμβάνει τη δρομολόγηση των δεδομένων φωνής. Στους δύο προσωπικούς υπολογιστές θα πρέπει να εκτελείται το υποσύστημα διεπαφής, ούτως ώστε οι χρήστες να μπορούν να χρησιμοποιήσουν τις λειτουργίες του προγράμματος. Όμως, μιας και πρόκειται για δύο διαφορετικούς υπολογιστές, μιλάμε για δύο διαφορετικές εκδοχές (στιγμιότυπα, εκφάνσεις) του υποσυστήματος αυτού. Ακόμα πιο χαρακτηριστική είναι η περίπτωση του υποσυστήματος επικοινωνίας. Μιας και όλοι ανεξαιρέτως οι κόμβοι θα πρέπει να δέχονται και να στέλνουν φωνητικά δεδομένα, εκφάνσεις του υποσυστήματος αυτού θα βρίσκονται σε λειτουργία τόσο στους προσωπικούς υπολογιστές των τελικών χρηστών όσο και στον ενδιάμεσο διακομιστή παρά τις διαφορές που μπορεί να έχουν από άποψη υλικού ή άλλων εκτελούμενων συστατικών στοιχείων, υποσυστημάτων ή λειτουργιών.



## Δραστηριότητα 5/Κεφάλαιο 9

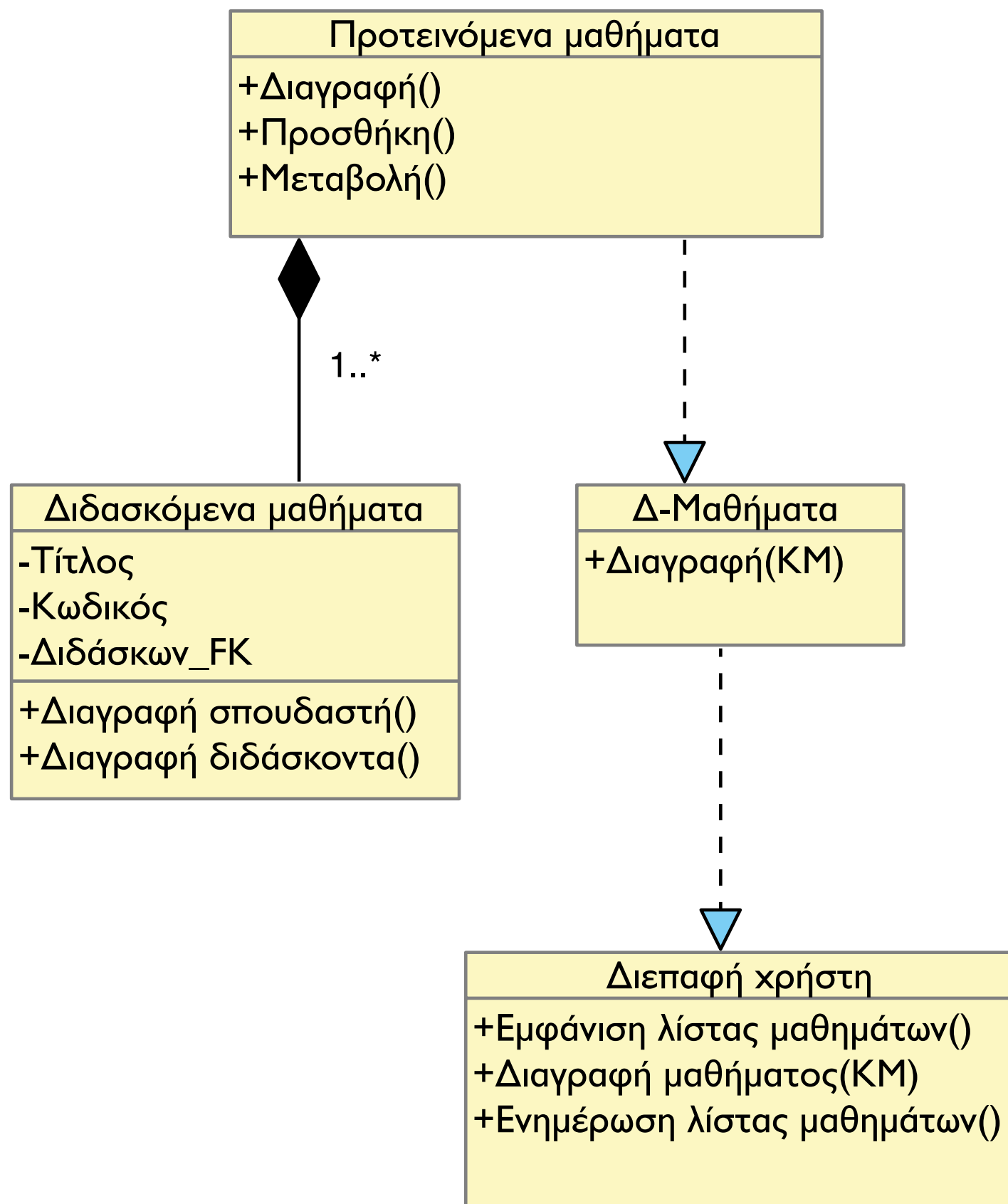
---

Υποθέτουμε πως η περίπτωση χρήσης «διαγραφή μαθήματος» έχει την ακόλουθη βασική ροή γεγονότων:

1. Ο χειριστής «χειριστής γραμματείας» επιλέγει από το μενού την εντολή «προβολή μαθημάτων».
2. Ο χειριστής επιλέγει ένα μάθημα από τη λίστα.
3. Ο χειριστής επιλέγει από το μενού την εντολή «διαγραφή μαθήματος».
4. Ο «Επίκουρος» ελέγχει αν υπάρχουν εγγεγραμμένοι σπουδαστές στο συγκεκριμένο μάθημα. Αν υπάρχουν, τους αποσυνδέει από τη λίστα σπουδαστών του μαθήματος.
5. Ο «Επίκουρος» ελέγχει αν υπάρχουν εγγεγραμμένοι καθηγητές που διδάσκουν το επιλεγμένο μάθημα. Αν υπάρχουν, τους αποσυνδέει από τη λίστα καθηγητών του μαθήματος.
6. Ο «Επίκουρος» διαγράφει μόνιμα το μάθημα.
7. Ο «Επίκουρος» ενημερώνει τη λίστα μαθημάτων του χρήστη.

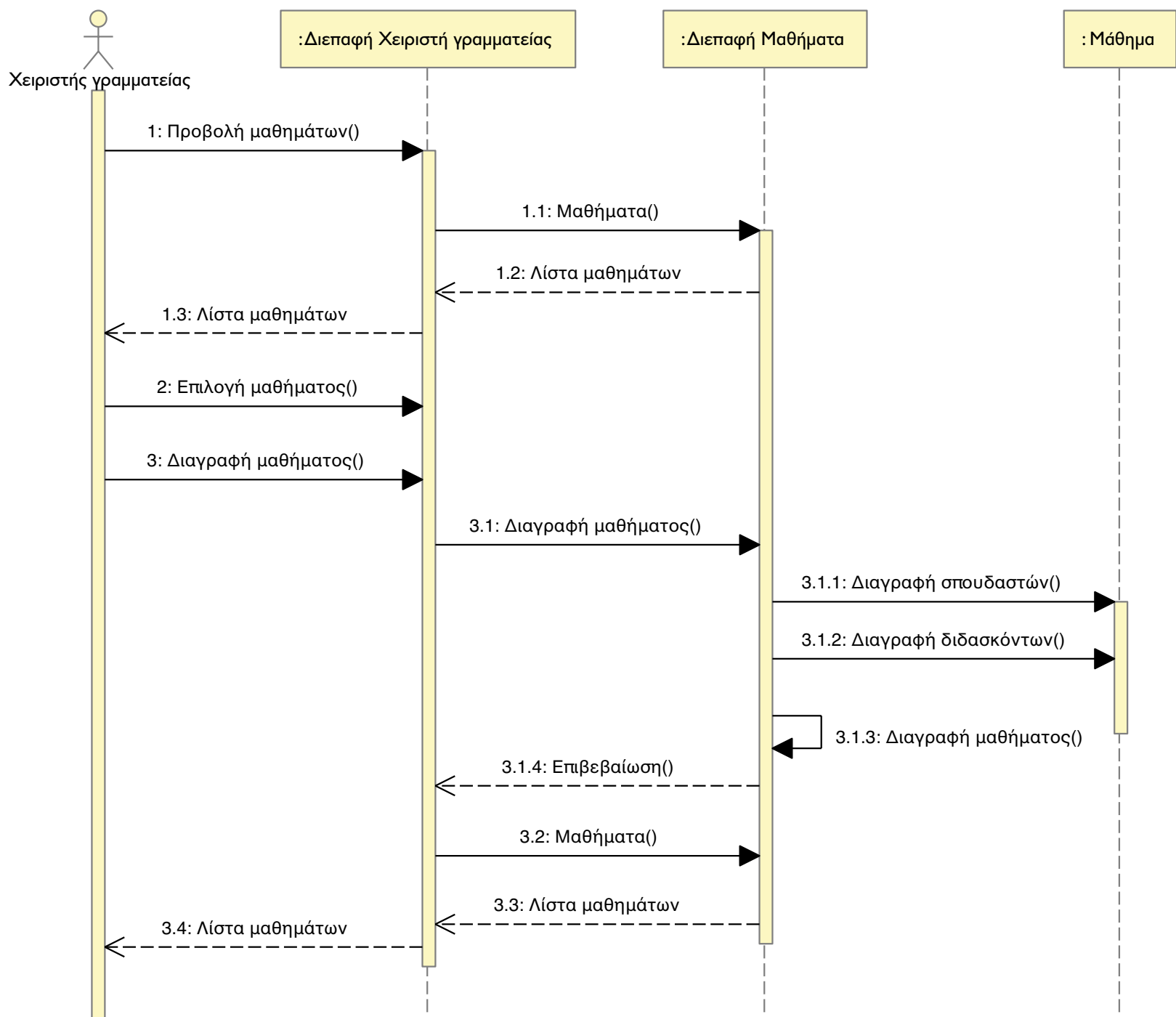
Βάσει αυτής της ροής και του διαγράμματος κλάσεων της περίπτωσης που μελετήσαμε πιο πάνω, μπορούμε να καταλήξουμε στο παρακάτω διάγραμμα κλάσεων.

**Σχήμα 9.15** Το ζητούμενο διάγραμμα κλάσεων.



Βέβαια, είναι κατανοητό πως αυτό το διάγραμμα θα τεθεί υπό αίρεση αν δούμε πως δεν είναι συνεπές με το αντίστοιχο διάγραμμα ακολουθίας. Οπότε, περνάμε στο διάγραμμα ακολουθίας για την προαναφερθείσα περίπτωση χρήσης:

**Σχήμα 9.16** Το ζητούμενο διάγραμμα ακολουθίας.



Αυτό το διάγραμμα ακολουθίας είναι επαρκές για τη συγκεκριμένη περίπτωση χρήσης. Επιστρέφοντας στο σχετικό διάγραμμα κλάσεων, παρατηρούμε ότι είναι συνεπές, με μόνη εξαίρεση την απουσία μιας μεθόδου στην κλάση «μαθήματα» η οποία να επιστρέφει μια λίστα μαθημάτων για παρουσίαση στον χειριστή, την οποία μπορείτε να προσθέσετε μόνοι σας.

## Άσκηση 1/Κεφάλαιο 9

---

Όπως αναφέρθηκε, τόσο στο μοντέλο ανάλυσης όσο και στο μοντέλο σχεδίασης τα δομικά στοιχεία είναι οι κλάσεις. Με αυτό κατά νου, δεν έχει νόημα να ισχυριστούμε ότι «η ανάλυση τελείωσε εδώ και αυτά είναι τα αποτελέσματά της», διότι αυτό απλά θα ήταν η καταγραφή ενός ανεπίκαιρου στιγμιότυπου κάποιων κλάσεων οι οποίες στη συνέχεια θα υποστούν μεταβολές. Η αντικειμενοστρεφής λογική χαρακτηρίζεται από συνέχεια και οι φάσεις είναι διακριτές όχι γιατί τα αποτελέσματά τους χαρακτηρίζονται κάποια στιγμή τετελεσμένα, αλλά γιατί καθεμία προσθέτει στο οικοδόμημα και από κάτι, μέχρι αυτό να χαρακτηριστεί πλήρες.

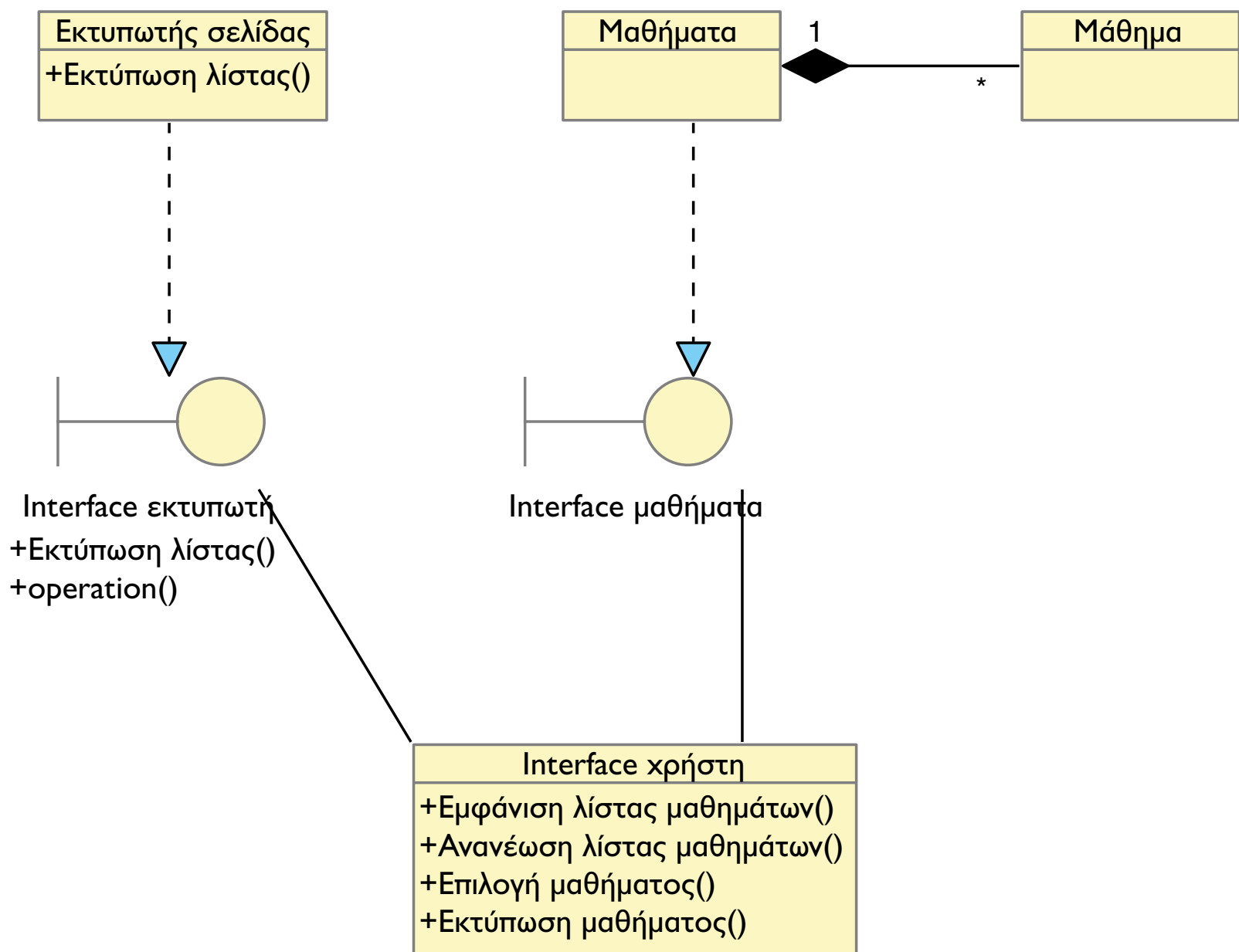
Οι φάσεις, δηλαδή, δεν οριοθετούνται πρωτίστως από τα αποτελέσματά τους, αλλά από την επίδρασή τους στα ίδια, κατ' ουσίαν, συστατικά στοιχεία λογισμικού τα οποία εμπλουτίζουν, εξειδικεύουν, κάνουν πιο συγκεκριμένα κ.λπ. Η ίδια η φύση των συστατικών στοιχείων λογισμικού επί των οποίων εργαζόμαστε επιτρέπει και ενθαρρύνει τη συμπεριφορά αυτή. Αυτή είναι η σημαντικότερη διαφορά από τη δομημένη ανάλυση και σχεδίαση, και ένας από τους κύριους λόγους διάδοσης της αντικειμενοστρεφούς τεχνολογίας.

## Άσκηση 2/Κεφάλαιο 9

---

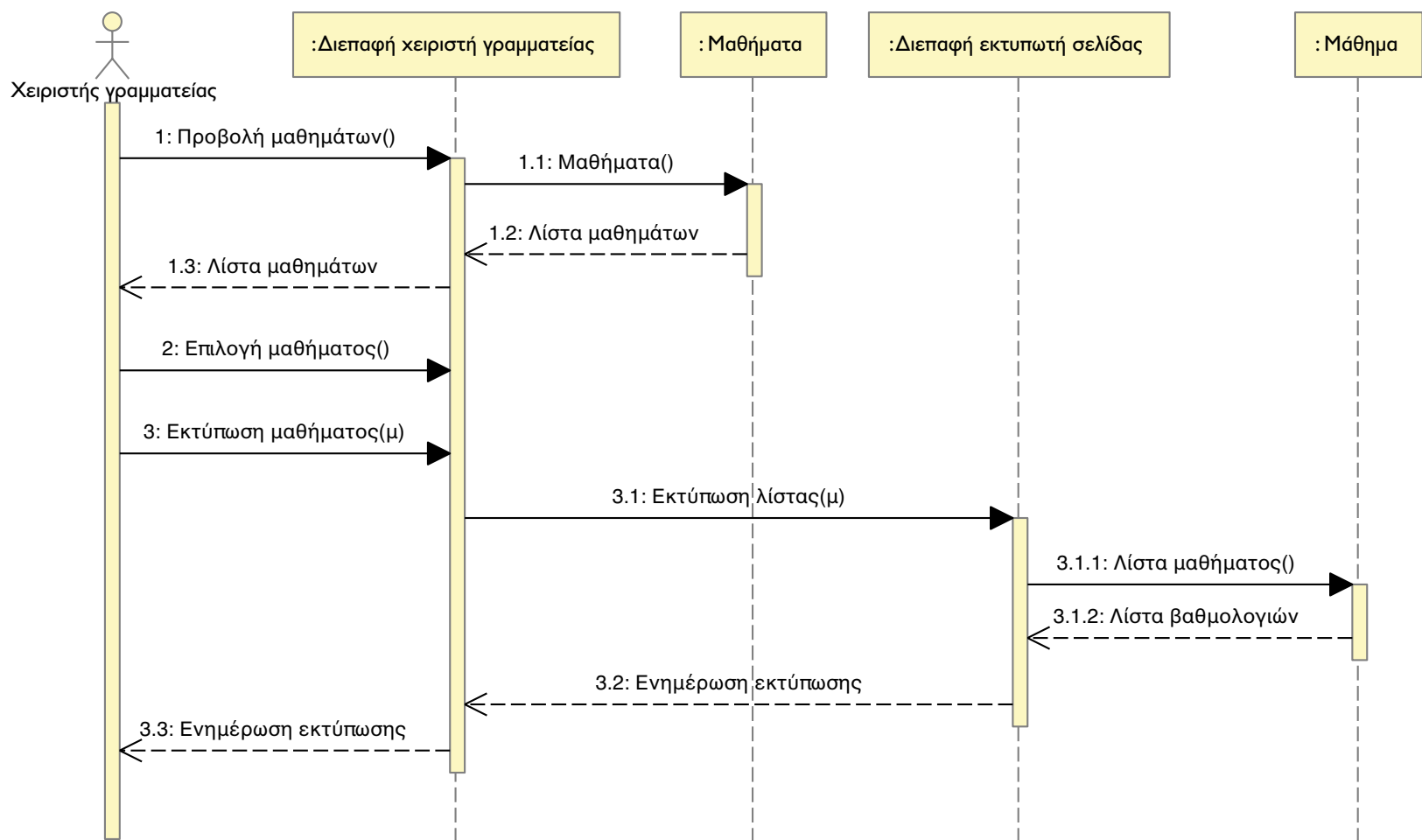
Το διάγραμμα κλάσεων φαίνεται στο σχήμα που ακολουθεί.

**Σχήμα 9.17** Το ζητούμενο διάγραμμα κλάσεων.



Το ζητούμενο διάγραμμα ακολουθίας φαίνεται στο σχήμα που ακολουθεί.

Σχήμα 9.18 Το ζητούμενο διάγραμμα ακολουθίας.





### Άσκηση 3/Κεφάλαιο 9

---

Προκειμένου να προσθέσουμε άλλο ένα κανάλι χρήσης του «Επίκουρου» μέσω κινητού τηλεφώνου, θα χρειαζόμασταν μια διαδικτυακή εφαρμογή προσαρμοσμένη σε κινητά τηλέφωνα, παρεμφερή με τον «διαδικτυακό Επίκουρο», η οποία να εκτελείται μέσα σε έναν καινούριο κόμβο, έκφανση του κινητού τηλεφώνου. Ο νέος κόμβος θα συνδέεται στον διακομιστή μέσω δικτύου δεδομένων κινητής τηλεφωνίας, για παράδειγμα, και το νέο υποσύστημα «κινητός Επίκουρος» θα επικοινωνεί όπως και ο «διαδικτυακός Επίκουρος» με την υπηρεσία « παρακολούθηση εκπαιδευτικής διαδικασίας». Εάν η υπηρεσία «παρακολούθηση εκπαιδευτικής διαδικασίας» βρισκόταν στους επιμέρους υπολογιστές, θα είχαμε μια διαφορετική διάταξη. Σε αυτούς τους κόμβους, τα υποσυστήματα διεπαφής θα επικοινωνούσαν με την «παρακολούθηση εκπαιδευτικής διαδικασίας» και αυτό το υποσύστημα, με τη σειρά του, με τον κεντρικό διακομιστή.

## ΒΙΒΛΙΟΓΡΑΦΙΑ

*Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example*, Doug Rosenberg και Kendall Scott, Addison-Wesley, ISBN 978-0201730395

Booch G., Rumbaugh J., Jacobson I., *The Unified Modeling Language User Guide*, Addison-Wesley.

Booch, G., *Object-Oriented Analysis and Design with Applications*, Addison-Wesley.

Fowler M., Scott K., *UML Distilled*, Addison-Wesley.

Jacobson I., Booch G., Rumbaugh J., *The Unified Software Development Process*, Addison-Wesley.

Jacobson I., Christerson M., Johnson P., Overgaard G., *Object-Oriented Software Engineering: A Use Case Driven Approach*, Addison-Wesley.

Pressman, R. S., *Software Engineering-A Practitioners Approach*, McGraw-Hill.

Quadrani T., *Visual Modeling with Rational Rose and UML*, Addison-Wesley.

Rumbaugh J., Jacobson I., Booch G., *The Unified Modeling Language Reference Manual*, Addison-Wesley.

Rumbaugh J., *Object-Oriented Modeling and Design*, Prentice Hall.

Sommerville, I. *Software Engineering*, London: Addison-Wesley.

## ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Σκοπός του κεφαλαίου είναι να εισάγει τον αναγνώστη στην υλοποίηση και τον έλεγχο λογισμικού σύμφωνα με την αντικειμενοστρεφή φιλοσοφία, συνεχίζοντας την αναφορά στην ενοποιημένη προσέγγιση ανάπτυξης λογισμικού.

Μετά τη μελέτη του κεφαλαίου, ο αναγνώστης θα είναι σε θέση:

- να κατασκευάζει ένα μοντέλο υλοποίησης μιας εφαρμογής λογισμικού σύμφωνα με την ενοποιημένη προσέγγιση ανάπτυξης λογισμικού, το οποίο αποτελείται από υποσυστήματα που περιέχουν συστατικά στοιχεία λογισμικού και διεπαφές,
- να υλοποιεί κλάσεις οι οποίες συμπεριλαμβάνονται στο μοντέλο υλοποίησης μιας εφαρμογής σύμφωνα με τον σχεδιασμό τους, που περιέχεται στο μοντέλο σχεδίασης,
- να προγραμματίζει και να πραγματοποιεί τον έλεγχο της υλοποίησης μιας εφαρμογής σύμφωνα με την ενοποιημένη προσέγγιση ανάπτυξης λογισμικού που βασίζεται στην αντικειμενοστρεφή τεχνολογία.

### Έννοιες-κλειδιά

---

- Μοντέλο υλοποίησης
- Συστατικό λογισμικού (*component*)
- Υποσύστημα
- Διεπαφή (*interface*)

Η εργασία που έπεται της σχεδίασης είναι η υλοποίηση, δηλαδή η συγγραφή του πηγαίου κώδικα του προγράμματος. Το βάρος, πλέον, δεν πέφτει στην κατασκευή μοντέλων, και τα προβλήματα που αντιμετωπίζονται δεν σχετίζονται με την εκφραστικότητα των μοντέλων, τον βαθμό αφαίρεσης, την κουλτούρα του σχεδιαστή και άλλες αφηρημένες έννοιες. Στη φάση της υλοποίησης αντιμετωπίζονται συγκεκριμένα κατασκευαστικά προβλήματα ή, ακριβέστερα, «προγραμματιστικά», όπως διαχείριση τύπων, επιλογή και χρήση προγραμματιστικών βιβλιοθηκών και πλαισίων προγραμματισμού κάθε είδους (APIs - Application Programming Interfaces), διαχείριση γεγονότων (events), εντοπισμός και διόρθωση σφαλμάτων (debugging) και αρκετά άλλα, γνωστά σε κάθε προγραμματιστή.

Η αφηρημένη αναφορά στο «πώς είναι το λογισμικό» (σχεδίαση), ώστε να «κάνει τα επιθυμητά» (ανάλυση), γίνεται πλέον συγκεκριμένη, γραμμή-προς-γραμμή υλοποίηση όσων έχουν καθοριστεί. Σε μια επιτυχή έκβαση ενός έργου ανάπτυξης λογισμικού, ο προγραμματιστής ξεκινώντας την υλοποίηση έχει αρκετά δεδομένα που θα τον καθοδηγήσουν να κάνει αυτό που πρέπει, ώστε το έργο να ολοκληρωθεί με επιτυχία, δηλαδή το λογισμικό να κάνει τις δουλειές για τις οποίες κατασκευάστηκε και να τις κάνει σωστά. Σε μια αποκλίνουσα από τα προσδοκώμενα έκβαση, ο προγραμματιστής, έχοντας στα χέρια του το σχέδιο του λογισμικού, μπορεί να ανατρέξει προς τα πίσω εντοπίζοντας τόσο στο πεδίο του συστήματος λογισμικού (σχεδίαση) όσο και στο πεδίο του προβλήματος (ανάλυση) το πρόβλημα. Αυτό, βέβαια, με την προϋπόθεση ότι έχει τηρηθεί μια ελάχιστη πειθαρχία κατά την κατασκευή των μοντέλων που αναφέρονται στα Κεφάλαια 8 και 9 του παρόντος βιβλίου.

Η δυνατότητα αυτή είναι ένα από τα σημαντικότερα πλεονεκτήματα της αντικειμενοστρεφούς τεχνολογίας και δεν δίνεται, τουλάχιστον σίγουρα όχι με την ίδια ευκολία, στη δομημένη ανάλυση και σχεδίαση. Εκεί, το να εντοπίσει κανείς σε ποια λειτουργική απαίτηση αντιστοιχεί μία μονάδα προγράμματος η οποία δεν μπορεί να κατασκευαστεί σωστά είναι πιο δύσκολο. Στην περίπτωση που εξετάζουμε εδώ, είναι σχεδόν άμεσο. Όσοι έχουν «πληρώσει το τίμημα» ενός αποτυχημένου έργου ανάπτυξης λογισμικού αντιλαμβάνονται ότι η κατά το δυνατόν πιστή εφαρμογή των βημάτων της αντικειμενοστρεφούς ανάλυσης και σχεδίασης και η δημιουργία όλων

των μοντέλων λογισμικού, μολονότι δεν οδηγεί σε εκτελέσιμο λογισμικό αλλά στην περιγραφή αυτού, αποδίδει τον χρόνο της και με το παραπάνω.

Το πρόβλημα της αναφοράς του θέματος σε ένα εγχειρίδιο όπως αυτό, χωρίς να γίνονται εκτενείς αναφορές σε ένα ή περισσότερα περιβάλλοντα ανάπτυξης, είναι μια μεγάλη πρόκληση. Επιλέξαμε στη συνέχεια του κεφαλαίου να χρησιμοποιήσουμε τη γλώσσα Java και μάλιστα χωρίς αναφορά σε βιβλιοθήκες προγραμματισμού περιβάλλοντος γραφικών ή και σε άλλα χαρακτηριστικά και δυνατότητες της γλώσσας, οι οποίες αφορούν συγκεκριμένα περιβάλλοντα λειτουργίας λογισμικού.

## ΕΝΟΤΗΤΑ 10.1. ΤΟ ΜΟΝΤΕΛΟ ΥΛΟΠΟΙΗΣΗΣ ΣΤΗΝ ΕΝΟΠΟΙΗΜΕΝΗ ΠΡΟΣΕΓΓΙΣΗ ΑΝΑΠΤΥΞΗΣ ΛΟΓΙΣΜΙΚΟΥ

Με την ολοκλήρωση της διαδικασίας της σχεδίασης, ο μηχανικός λογισμικού πρέπει να έχει στα χέρια του μια πλήρως τεκμηριωμένη λύση για το αρχικό πρόβλημα που επιδιώκει να λύσει. Έχουμε ήδη δει με ποιο τρόπο κατά τη σχεδίαση το πεδίο δράσης μας μετατοπίζεται από τη μοντελοποίηση του αρχικού προβλήματος στη μοντελοποίηση κάποιας αποδεκτής λύσης του. Η υλοποίηση αποτελεί το τελευταίο στάδιο αυτού του κύκλου ανάπτυξης, τουλάχιστον ως προς τη σύνθεση προσεγγίσεων και λύσεων που θα μας οδηγήσουν στην τελική εφαρμογή, δηλαδή, με απλά λόγια, σε ένα πρόγραμμα που δουλεύει.

Κατά την υλοποίηση, ο προγραμματιστής καλείται να υλοποιήσει τη σχεδίαση της εφαρμογής σε κάποια γλώσσα προγραμματισμού (ή σε περισσότερες από μία γλώσσες για κάποια συστήματα), να εγκαταστήσει τα παραδοτέα στον προβλεπόμενο εξοπλισμό, σύμφωνα με την προσυμφωνημένη διάταξη (deployment), ολοκληρώνοντας έτσι το έργο. Λαμβάνοντας υπόψη μας πως η συμπεριφορά του συστήματος έχει οριστεί κατά τη σχεδίαση, ο προγραμματιστής κατά την υλοποίηση πρέπει να προχωρήσει σε χαμηλότερο επίπεδο, δηλαδή πιο κοντά στη μηχανή. Έτσι, εκτός της απλής και μηχανικής μεταφοράς της σχεδίασης σε κάτι απτό, ο προγραμματιστής θα πρέπει να αποφασίσει, με δεδομένα τα εργαλεία προγραμματισμού που θα



χρησιμοποιήσει, σχετικά με τις δομές δεδομένων και τους αλγόριθμους που θα κρίνει πως είναι οι καταλληλότεροι για την καλύτερη δυνατή υλοποίηση του συστήματος στο εκάστοτε περιβάλλον. Στην πράξη, η απόφαση για πολλά από αυτά δεν μπορεί να ληφθεί μόνο από τον προγραμματιστή, αλλά αποτελεί αρχική παραδοχή ολόκληρου του έργου ανάπτυξης λογισμικού, την οποία ο προγραμματιστής καλείται απλά να εφαρμόσει. Επίσης, η μεταφορά της σχεδίασης σε γλώσσα προγραμματισμού σπάνια είναι μηχανική διεργασία: έχει αρκετή δημιουργικότητα και αυτό επιβεβαιώνεται συνεχώς από τους «μαχόμενους» προγραμματιστές.

Όπως και στα προηγούμενα βήματα της ενοποιημένης προσέγγισης, έτσι και κατά την υλοποίηση είναι πιθανό να χρειαστεί να κινηθεί κάποιος οπισθοδρομώντας σε προηγούμενα στάδια, αν αυτό κριθεί απαραίτητο. Για παράδειγμα, μπορεί να προκύψει κάποιο πρόβλημα στην υλοποίηση, που να αναγκάσει την ομάδα της ανάπτυξης να επανεξετάσει κάποια σημεία της σχεδίασης προτού να μπορέσει να επιστρέψει στην υλοποίηση. Περαιτέρω, η διαδικασία της υλοποίησης είναι εξ ορισμού συνδεδεμένη με τον έλεγχο που πρέπει να υποστεί η εφαρμογή πριν από την παράδοση σε λειτουργία.

### **10.1.1 Συστατικό λογισμικού (Component) στο μοντέλο υλοποίησης**

Στην υλοποίηση, με τον όρο «συστατικό λογισμικού» αναφερόμαστε σε κάθε υλοποιημένο τμήμα της εφαρμογής που έχει κάποιον ρόλο στη συνολική λειτουργία του υποσυστήματος στο οποίο ανήκει. Έτσι, ένα συστατικό λογισμικού μπορεί να είναι μία υλοποιημένη κλάση ή μία ομάδα κλάσεων. Επιπρόσθετα, συστατικό λογισμικού μπορεί να αποτελέσει η υλοποιημένη διεπαφή μεταξύ του συστήματος που κατασκευάζουμε και ενός άλλου, εξωτερικού συστήματος λογισμικού όπως, για παράδειγμα, μιας βάσης δεδομένων.

Αναλόγως με τον τρόπο με τον οποίο έχουν οριστεί διάφορα συστατικά λογισμικού κατά τη σχεδίαση, προχωρούμε και στη δημιουργία των αντίστοιχων κατά την υλοποίηση. Η έννοια του συστατικού λογισμικού ως ένα σύνολο κλάσεων ή άλλων προγραμματιστικών στοιχείων μπορεί να διατηρηθεί σε διάφορες γλώσσες προγραμματισμού, κάνοντας έτσι τη μετάβαση

από τη σχεδίαση στην υλοποίηση, και το αντίστροφο, πιο εύκολη. Για παράδειγμα, η Java παρέχει στον προγραμματιστή την έννοια του πακέτου (package). Τα πακέτα της Java μπορούν να χρησιμοποιηθούν για να ορίσουν συστατικά λογισμικού με ιεραρχικό τρόπο. Στο επίπεδο του λειτουργικού συστήματος, αυτά τα πακέτα εκφράζονται με τη δημιουργία φακέλων. Έτσι, κλάσεις και διεπαφές που βρίσκονται σε κάποιον φάκελο ανήκουν και στο ίδιο συστατικό λογισμικού κ.ο.κ.

Είναι σκόπιμο να τονίσουμε ότι σε συστήματα λογισμικού μικρής έκτασης, όπως είναι τα παραδείγματα αυτού του βιβλίου, ένα συστατικό λογισμικού μπορεί να είναι ταυτόσημο με μία κλάση.

## ΠΑΡΑΔΕΙΓΜΑ

**ΣΥΝΤΑΞΗ JAVA:** Ο τρόπος με τον οποίο ορίζονται τα συνθετήματα ή πακέτα στη γλώσσα προγραμματισμού Java είναι με τη λέξη-κλειδί (keyword) `package`, ακολουθούμενο από το όνομα του πακέτου στην αρχή του σχετικού αρχείου πηγαίου κώδικα. Εσωτερικά πακέτα διαχωρίζονται με μία τελεία (.). Ας υποθέσουμε πως έχουμε ένα συνθέτημα που υλοποιεί μία διεπαφή με τους χρήστες, μια φόρμα, δηλαδή, που αποτελείται από κάποια στοιχεία για την απεικόνιση δεδομένων (έξοδος) και κάποια για την εισαγωγή δεδομένων από τους χρήστες (είσοδος). Η φόρμα αυτή θα ήταν λογικό να ανήκει σε κάποιο πακέτο σχετικό με τη διεπαφή με τους χρήστες. Ονομάζουμε αυτό το πακέτο «ui» (για user interface). Στην αρχή του αρχείου του πηγαίου κώδικα της φόρμας δηλώνουμε πως η κλάση της φόρμας ανήκει στο πακέτο «ui», γράφοντας:

```
package ui;
```

```
class InputForm
```

```
{
```

```
    // η υλοποίηση της κλάσης θα γίνει εδώ
```

```
}
```

Τα στοιχεία της φόρμας που βοηθούν στην είσοδο των δεδομένων (όπως κουμπιά, μενού κ.λπ.) μπορούμε να τα τοποθετήσουμε σε ένα άλλο πακέτο, το οποίο όμως να συμπεριλαμβάνεται στο «ui» ως εξής:

```
package ui.input;

class MyButton
{
    // η υλοποίηση της κλάσης θα γίνει εδώ
}
```

Με αυτόν τον τρόπο μπορούμε να επιτύχουμε την ιεράρχηση ολόκληρης της εφαρμογής ή επιμέρους υποσυστημάτων της σε πακέτα ή συνθετήματα. Αυτή η διαίρεση δεν επηρεάζει ουσιαστικά την υλοποίηση, όμως αποτελεί καλή πρακτική έτσι ώστε να είναι πιο εύκολη και κατανοητή η μετάβαση από τη σχεδίαση στην υλοποίηση. Επίσης, μας παρέχει μοναδικές ονομασίες που χρησιμεύουν ως σημεία αναφοράς για τα διάφορα συνθετήματα και υποσυστήματα. Τέτοια σημεία αναφοράς είναι σημαντικά τόσο για τη σχεδίαση και την υλοποίηση όσο και για τον έλεγχο, όπως θα δούμε παρακάτω.

### 10.1.3 Υποσύστημα στο μοντέλο υλοποίησης

Ένα υποσύστημα είναι ένα σύνολο από συστατικά λογισμικού. Δεν είναι σαφές, ούτε και εντός της φιλοσοφίας της ενοποιημένης προσέγγισης ανάπτυξης λογισμικού, να καθοριστεί επακριβώς τι είναι ένα υποσύστημα. Οι εκάστοτε συνθήκες οδηγούν πάντα τον έμπειρο μηχανικό λογισμικού σε σωστές αποφάσεις. Καταφεύγουμε σε ένα παράδειγμα, στο οποίο αναπόφευκτα κάνουμε χρήση ονομάτων συγκεκριμένων εφαρμογών λογισμικού:

Αν, για παράδειγμα, ως σύστημα εννοούμε ένα σύνολο εφαρμογών γραφείου όπως, λόγου χάρη, το δωρεάν διατιθέμενο λογισμικό LibreOffice, τότε τα υποσυστήματά του είναι οι επιμέρους εφαρμογές που ονομάζονται



«Writer», «Calc», «Impress» και «Draw». Κάθε τέτοιο υποσύστημα περιέχει πολλά συστατικά λογισμικού (modules) και κάθε συστατικό έχει πολλές κλάσεις.

Ωστόσο, αν ως σύστημα θεωρήσουμε το παράδειγμα «Επίκουρος» του παρόντος βιβλίου, θα δυσκολευτούμε να ορίσουμε υποσυστήματα τα οποία να μην περιέχουν μόνο ένα συστατικό λογισμικού, το οποίο με τη σειρά του να περιέχει μόνο μία κλάση. Ο κίνδυνος παρανοήσεων λόγω της μικρής έκτασης κάθε εκπαιδευτικού παραδείγματος είναι υπαρκτός, όμως, έχοντας κατά νου το παραπάνω παράδειγμα, οι παρανοήσεις περιορίζονται.

#### **10.1.4 Διεπαφές στο μοντέλο υλοποίησης**

Όπως έχουμε δει στο προηγούμενο κεφάλαιο (Ενότητα 9.2.3), οι διεπαφές (interfaces) είναι πολύ σημαντικές για την επιτυχή ανάπτυξη μιας εφαρμογής λογισμικού. Με τη χρήση διεπαφών μπορούμε να ορίσουμε τον τρόπο επικοινωνίας διαφόρων μερών του λογισμικού όπως επίσης και τον τρόπο επικοινωνίας μερών του λογισμικού με εξωτερικά συστήματα. Είναι σημαντικό να μην ταυτίζουμε την έννοια «διεπαφή» (η οποία αποδίδει γενικά τον όρο «interface») με το «user interface» –η επικοινωνία με τον χρήστη δεν είναι η μόνη αλληλεπίδραση του λογισμικού με τον «έξω κόσμο». Επιπλέον, με τη χρήση διεπαφών μπορούμε να επιτρέψουμε την προσθήκη εναλλακτικών υλοποιήσεων και τη δυναμική φόρτωσή τους κατά τον χρόνο εκτέλεσης της εφαρμογής, χαρακτηριστικό εγγενές στην αντικειμενοστρεφή τεχνολογία. Τέλος, οι διεπαφές μάς διευκολύνουν στο να συνδέσουμε μονάδες εισόδου – εξόδου με τα κεντρικά υποσυστήματα μιας εφαρμογής αποφεύγοντας ανώφελες απευθείας ζεύξεις μεταξύ τους, που μπορεί να δημιουργήσουν προβλήματα κατά τη διόρθωση σφαλμάτων όπως επίσης και κατά τη συντήρηση της εφαρμογής.

Η υλοποίηση των διεπαφών είναι ένα από τα πρώτα μελήματα του προγραμματιστή. Ξεκινώντας από τις διεπαφές επιτυγχάνουμε την υλοποίηση του σκελετού της εφαρμογής, καλύπτοντας τα κομβικά σημεία σε σύντομο χρονικό διάστημα. Μετά από την ολοκλήρωση αυτής της εργασίας, μπορούμε να συνεχίσουμε με την υλοποίηση αλγορίθμων και ολοκληρωμένων

κλάσεων, των οποίων οι εξωτερικές μέθοδοι είναι άρτια ορισμένες από τις αντίστοιχες διεπαφές. Στις περισσότερες γλώσσες προγραμματισμού οι διεπαφές αποτελούν αναπόσπαστο μέρος τους και υποστηρίζονται από δεσμευμένες λέξεις-κλειδιά (reserved words).

## ΠΑΡΑΔΕΙΓΜΑ

Στη Java, ο ορισμός των διεπαφών γίνεται με τη χρήση της λέξης-κλειδιού `interface`. Στη συγκεκριμένη γλώσσα προγραμματισμού οι διεπαφές βρίσκονται στην κορυφή των επιτρεπόμενων ταξινομιών κλάσεων μέσω των διαδικασιών της υλοποίησης ή της κληρονομικότητας. Με άλλα λόγια, μία κλάση μπορεί να υλοποιήσει μία διεπαφή ή/και να κληρονομήσει μία άλλη κλάση, αλλά ποτέ το αντίστροφο. Επίσης, μία κλάση μπορεί να υλοποιεί περισσότερες από μία διεπαφές. Τέλος, είναι επιτρεπτό για μία διεπαφή να κληρονομεί μία άλλη διεπαφή.

Ας αναλογιστούμε την εφαρμογή «Επίκουρος». Στη δραστηριότητα I του προηγούμενου κεφαλαίου, είδαμε πως το πρόγραμμα αυτό θα μπορούσε να παρέχει και την υπηρεσία της εκτύπωσης διαφόρων δεδομένων σε διάφορα μέσα. Για τον λόγο αυτό καταλήξαμε σε μία διεπαφή «εκτυπωτής», η οποία θα πρέπει να παρέχει τις απαραίτητες μεθόδους που θα καλεί η εφαρμογή έτσι ώστε να μπορεί να εκτυπώσει. Όσον αφορά το κεντρικό παράθυρο της εφαρμογής (για παράδειγμα, αυτό στο οποίο θα εργάζεται η γραμματεία), ο κάθε εκτυπωτής θα πρέπει να έχει μία μέθοδο στην οποία η γραμματεία θα δίνει μέσω μιας παραμέτρου έναν σπουδαστή (ένα αντικείμενο της κλάσης «σπουδαστής»), έτσι ώστε να γίνεται η εκτύπωση, σύμφωνα με τον τύπο του εν λόγω εκτυπωτή. Μια τέτοια διεπαφή φαίνεται παρακάτω:

```
public interface PrinterI
{
    public String printerIdentification();
    public void printStudent(Student someStudent);
}
```

Σύμφωνα με αυτήν τη διεπαφή, κάθε υλοποίηση κλάσης εκτυπωτή θα πρέπει να περιέχει την υλοποίηση των δύο μεθόδων. Συμπεριλαμβάνουμε και μία μέθοδο «printerIdentification», έτσι ώστε το κεντρικό παράθυρο της εφαρμογής να γνωρίζει σε ποιον εκτυπωτή αναθέτει την εκτύπωση.

Τώρα, ας υποθέσουμε πως στην εφαρμογή μας θέλουμε να συμπεριλάβουμε έναν νέο εκτυπωτή, που θα εκτυπώνει τα στοιχεία ενός σπουδαστή στην οθόνη μας. Η κλάση που θα μας δώσει αυτήν τη λειτουργία θα πρέπει να υλοποιεί την παραπάνω διεπαφή, μιας και η επικοινωνία του κεντρικού παραθύρου της εφαρμογής θα γίνεται αποκλειστικά μέσω αυτής (της διεπαφής). Στην Java, μια τέτοια κλάση υλοποιείται ως εξής:

```
public class MonitorPrinter implements PrinterI
{
    public String printerIdentification() {
        return "monitor printer";
    }

    public void printStudent(Student someStudent)
    {
        System.out.println(someStudent.firstName);
        System.out.println(someStudent.lastName);
        // επιπλέον πληροφορίες του σπουδαστή
    }
}
```

Όταν μία κλάση όπως η παραπάνω υλοποιεί (implements) μία διεπαφή, είναι αναγκαστικό να υλοποιήσει όλες τις μεθόδους που προβλέπονται από τη διεπαφή αυτή.

Ο κώδικας πίσω από το κεντρικό παράθυρο της εφαρμογής, προκειμένου να εκτυπωθεί το αρχείο ενός σπουδαστή, θα είναι μόνο μία κλήση στη μέθοδο της διεπαφής «printStudent». Έτσι, στην περίπτωση που είχαμε έναν

επιπλέον εκτυπωτή, για παράδειγμα «PaperPrinter», η κεντρική εφαρμογή θα έβλεπε μόνο τη διεπαφή, και ο ίδιος κώδικάς της που θα εκτύπωνε σε χαρτί, θα εκτύπωνε και στην οθόνη. Αυτό, βέβαια, θα συνέβαινε δεδομένου ότι τα αντίστοιχα αντικείμενα των «PaperPrinter» και «MonitorPrinter» είχαν δημιουργηθεί.

Από το παράδειγμα αυτό είναι φανερό πώς οι διεπαφές βοηθούν στην υλοποίηση σύνθετων λύσεων μέσω της ευελιξίας που παρέχουν στον προγραμματιστή.

## Σύνοψη ενότητας

---

*Το μοντέλο υλοποίησης περιέχει τα εκτελέσιμα στοιχεία του λογισμικού, οργανωμένα (από κάτω προς τα πάνω) σε κλάσεις, συστατικά στοιχεία (components), υποσυστήματα και συστήματα. Στην περίπτωση που μια εφαρμογή λογισμικού δεν είναι εκφυλιστικά απλή, τότε οι έννοιες «σύστημα», «υποσύστημα», «component» και «κλάση» δεν ταυτίζονται. Σε όλες τις περιπτώσεις, οι διεπαφές (interfaces) περιγράφουν την επικοινωνία ενός συστατικού με το περιβάλλον του και υποστηρίζονται από όλες τις σύγχρονες γλώσσες προγραμματισμού.*

## ΕΝΟΤΗΤΑ 10.2. ΕΡΓΑΣΙΕΣ ΠΟΥ ΕΚΤΕΛΟΥΝΤΑΙ ΣΤΗΝ ΥΛΟΠΟΙΗΣΗ

Η γενική ροή εργασιών που πρέπει να εκτελεστούν κατά την υλοποίηση είναι η ακόλουθη:

1. Υλοποίηση αρχιτεκτονικής του λογισμικού.
2. Υλοποίηση συστημάτων – υποσυστημάτων.
3. Υλοποίηση κλάσεων.

Σύμφωνα με την αναφορά στην έννοια του συστήματος – υποσυστήματος η οποία προηγήθηκε, μια μικρής έκτασης εφαρμογή λογισμικού, όπως τα παραδείγματα που μας απασχολούν εδώ, έχει μόνο υλοποίηση κλάσεων. Σε μεγαλύτερες εφαρμογές, όπου τα συστήματα και τα υποσυστήματα δεν ταυτίζονται με κλάσεις, έχει νόημα να μιλάμε για υλοποίηση αυτών. Έχοντας αναφέρει τα παραπάνω, η εργασία που αναφέρεται ως υλοποίηση συστημάτων – υποσυστημάτων έχει ακριβώς την έκταση που της προσδίδει η ανάλυση του λογισμικού μας σε συστήματα και υποσυστήματα.

Μια ευρύτερη έννοια είναι η υλοποίηση της αρχιτεκτονικής, η οποία έχει νόημα στις περιπτώσεις που το λογισμικό ακολουθεί κάποιο από τα μοντέλα διάταξης στα οποία έγινε αναφορά στο Κεφάλαιο 3. Κατά την υλοποίηση της αρχιτεκτονικής ακολουθούμε όλα τα βήματα της υλοποίησης συστημάτων, υποσυστημάτων, κλάσεων για καθένα στοιχείο (tier) της αρχιτεκτονικής. Είναι φανερό ότι η καρδιά της υλοποίησης είναι η εργασία της υλοποίησης των κλάσεων, στην οποία θα αναφερθούμε ευθύς αμέσως.

### Υλοποίηση κλάσης

Οι κλάσεις είναι το κεντρικό δομικό στοιχείο της αντικειμενοστρεφούς ανάπτυξης λογισμικού. Επομένως, η όποια παρεχόμενη λειτουργία μιας εφαρμογής επιτυγχάνεται από την υλοποίηση κλάσεων. Έχουμε ήδη δει πώς οι διεπαφές βοηθούν στην ανάπτυξη ευέλικτων συστημάτων, όμως η υλοποίηση των κλάσεων ενός σχεδίου είναι η διαδικασία που εν τέλει θα μας δώσει το τελικό προϊόν.

Η μετάβαση από την κλάση της σχεδίασης στην υλοποίησή της είναι απλή στη θεωρία αλλά, πολλές φορές, απαιτητική στην πράξη. Οι δυσκολίες υλοποίησης εντοπίζονται κυρίως στον καθορισμό κατάλληλων δομών

δεδομένων και αλγόριθμων ικανών να υποστηρίξουν τις επιθυμητές λειτουργίες της κλάσης. Ο προγραμματιστής, επομένως, καλείται να υλοποιήσει όσα έχουν προηγουμένως περιγραφεί στα διαγράμματα κλάσεων και αλληλουχίας (sequence), τα οποία όμως περιορίζονται στο να περιγράψουν τη συμπεριφορά που πρέπει να επιτευχθεί και όχι τον ακριβή τρόπο με τον οποίο θα μπορούσε αυτή να επιτευχθεί. Σε κάθε περίπτωση, καλό είναι ο προγραμματιστής να ξεκινά με τα στοιχεία που έχει από τη σχεδίαση και να εργάζεται έτσι ώστε να καλύψει σταδιακά τις λειτουργικές απαιτήσεις της κάθε κλάσης. Εργαζόμενος προς αυτήν την κατεύθυνση, είναι πιθανό να ανακαλύψει κανείς αδυναμίες στη σχεδίαση. Σε αυτήν την περίπτωση θα πρέπει να κινηθεί αναδρομικά, διορθώνοντας τη σχεδίαση, προτού επιστέψει στη συνέχεια της υλοποίησης.

Η υλοποίηση της κλάσης είναι συνδεδεμένη με τον έλεγχό της. Στο Κεφάλαιο II θα συζητήσουμε τον έλεγχο μονάδας (unit testing). Αυτός ο τύπος ελέγχου αναφέρεται στον επαυξητικό έλεγχο της κλάσης κατά τη διάρκεια της υλοποίησής της. Ο προγραμματιστής που εφαρμόζει αυτόν τον έλεγχο μπορεί να υλοποιήσει την κλάση και στη συνέχεια να την ελέγξει για την ορθότητά της, να ελέγξει, δηλαδή, αν καλύπτει τις ανάγκες της σχεδίασης. Εναλλακτικά, θεωρείται καλή πρακτική ο προγραμματιστής να γράφει τους σχετικούς ελέγχους για μία κλάση προτού προχωρήσει στην υλοποίησή της. Η υλοποίηση τότε καθοδηγείται μόνο από το αν ο έλεγχος πραγματοποιείται επιτυχώς. Για την παρακάτω περίπτωση χρήσης δεν θα λάβουμε υπόψη μας την εφαρμογή ελέγχου μονάδας.

## ΜΕΛΕΤΗ ΠΕΡΙΠΤΩΣΗΣ

Ας επιστέψουμε στην εφαρμογή «Επίκουρος» και πιο συγκεκριμένα στην κλάση «σπουδαστές», όπως αυτή περιγράφεται στα Σχήματα 9.5 και 9.6 για την περίπτωση χρήσης «διαγραφή σπουδαστή». Θα δούμε μια ημιτελή υλοποίηση αυτής της κλάσης στη Java. Ο αναγνώστης καλείται να μελετήσει τις λεπτομέρειες της υλοποίησης στον δικό του χρόνο. Εξηγούμε πως η κλάση είναι ημιτελής μιας και θα υλοποιήσουμε μόνο τα μέρη που είναι σχετικά με τη διαγραφή φοιτητών.



Για την υλοποίηση ξεκινούμε από τις γνωστές μεθόδους που περιγράφονται από το διάγραμμα κλάσης. Από το σχετικό διάγραμμα αλληλουχίας παρατηρούμε ότι αυτή η κλάση δεν καλεί μεθόδους κάποιας άλλης κλάσης που θα είχαμε να υλοποιήσουμε. Άρα, αποτελεί ένα καλό σημείο εκκίνησης της υλοποίησης. Στην πραγματικότητα χρησιμοποιεί την κλάση «σπουδαστής», την οποία δεν θα συμπεριλάβουμε σε αυτήν τη μελέτη περίπτωσης. Ας υποθέσουμε, για χάρη του παραδείγματος, πως αποτελείται μόνο από δύο πεδία, «firstName» και «lastName», των οποίων τις τιμές μπορούμε να θέσουμε ή να ανακτήσουμε. Πριν ξεκινήσουμε την υλοποίηση της κλάσης παρατηρούμε πως στη σχεδίαση έχει προβλεφθεί μία λίστα σπουδαστών. Αποφασίζουμε να υλοποιήσουμε αυτήν τη λίστα ως μία απλή δυναμική λίστα. Η Java περιλαμβάνει μια τέτοια λίστα στη βιβλιοθήκη της που ονομάζεται «ArrayList». Άρα, το πεδίο αυτό της λίστας θα είναι ένα αντικείμενο της κλάσης «ArrayList» που παρέχει η Java. Μια αρχική υλοποίηση της κλάσης αυτής είναι η εξής:

```

import java.util.ArrayList;

public class Students implements StudentsI
{
    // Η λίστα με τους φοιτητές
    private ArrayList studentList = null;

    public Students()
    {
        // Αρχικοποίηση της λίστας φοιτητών
        studentList = new ArrayList();
    }

    public void removeStudent(Student s)
    {
        // Εύρεση της θέσης του φοιτητή στη λίστα
        int position = findPositionOfStudent(s);
        // Διαγραφή του
        studentList.remove(position);
    }

    public ArrayList difference(ArrayList stlist)
    {
        // Η λίστα που θα επιστραφεί από τη μέθοδο
        ArrayList diffList = new ArrayList();

        // Για κάθε φοιτητή στη δεδομένη λίστα ψάχνουμε στην
        // εσωτερική.
        // Αν δεν τον εντοπίσουμε τότε τον προσθέτουμε στη
        // λίστα που θα επιστρέψουμε
        for (int i = 0; i < stlist.size; i++)
        {
            Student s = (Student)stlist.get(i);
            int position = findPositionOfStudent(s);
            // αν η θέση του φοιτητή στη λίστα είναι < 0
            // τότε ο φοιτητής δε βρέθηκε...
            if (position < 0)
            {
                // ... και άρα πρέπει να προστεθεί:
                diffList.add(s);
            }
        }
        return diffList;
    }
}

```



Από την υλοποίηση προκύπτει πως και για τις δύο μεθόδους θα χρειαστεί να εντοπίσουμε μεμονωμένους σπουδαστές στην εσωτερική μας λίστα. Για τον λόγο αυτό χρησιμοποιούμε τη μέθοδο «`findPositionOfStudent(s: Student): int`». Αυτή η μέθοδος θα πρέπει να μας επιστρέψει τη θέση του σπουδαστή στην εσωτερική μας λίστα, που παίρνει τιμές από 0 έως  $n-1$ , όπου « $n$ » ο αριθμός των σπουδαστών στη λίστα. Αφού αυτή η μέθοδος χρησιμοποιείται εσωτερικά και δεν απαιτείται από καμία διεπαφή στη σχεδίαση, μπορούμε να την υλοποιήσουμε ως εσωτερική της κλάσης. Είναι φανερό πως η υλοποίησή της θα εξαρτηθεί από την απόφασή μας να χρησιμοποιήσουμε μια λίστα τύπου «`ArrayList`». Αν είχαμε επιλέξει μια διαφορετική δομή, η μέθοδος αυτή θα ήταν επίσης διαφορετική. Αυτός είναι ένας επιπλέον λόγος για να την τοποθετήσουμε σε μία άλλη, εσωτερική μέθοδο και όχι μέσα στις υπάρχουσες μεθόδους. Δεδομένης της παραπάνω απόφασής μας, μια υλοποίηση της μεθόδου «`findPositionOfStudent`» είναι η ακόλουθη:

```

private int findPositionOfStudent(Student s)
{
    int pos = -1;
    for (int i = 0; i < studentList.size; i++)
    {
        Student currentStudent = (Student)studentList.
get(i);
        // Ο σπουδαστής βρέθηκε αν και το μικρό του όνομα
        // και το επώνυμό του συμπίπτουν με κάποιου στην
        // εσωτερική λίστα.
        if (s.firstName.equals(currentStudent.firstName) &&
            s.lastName.equals(currentStudent.lastName))
        {
            pos = i;
            break;
        }
    }
    return pos;
}

```

Μια τέτοια υλοποίηση, αν και απέχει από το να είναι η πλέον αποδεκτή εξαιτίας αλγοριθμικών προβλημάτων, είναι πλήρως συμβατή με τη λειτουργική απαίτηση που προκύπτει από την εν λόγω περίπτωση χρήσης.

## Δραστηριότητα I/Κεφάλαιο I0

Στην παραπάνω υλοποίηση, ενώ δηλώνεται πως η κλάση μας υλοποιεί τη διεπαφή «Δ-Σπουδαστές» (StudentsI), η υλοποίησή της δεν δίνεται. Υλοποιήστε τη διεπαφή και συνεχίστε την υλοποίηση της κλάσης «σπουδαστές» (Students), έτσι ώστε να υποστηρίζει και την προσθήκη νέου σπουδαστή.

## Άσκηση I/Κεφάλαιο I0

Στην παραπάνω μελέτη περίπτωσης αναφέραμε πως η υλοποίηση δεν είναι η καλύτερη δυνατή από αλγοριθμικής άποψης. Ποια προβλήματα εντοπίζετε εξαιτίας της χρήσης της κλάσης «ArrayList» για τη συγκεκριμένη λειτουργία; Ποια εναλλακτική δομή δεδομένων θα μπορούσαμε να χρησιμοποιήσουμε για να περιορίσουμε τα προβλήματα αυτά;

## ΕΝΟΤΗΤΑ 10.3. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ

### Δραστηριότητα I/Κεφάλαιο I0

Η διεπαφή «Δ-Σπουδαστές», στην οποία έχουμε αναφερθεί και στη μελέτη περίπτωσης στην Ενότητα 9.3.1, θα πρέπει να ορίζει μεθόδους μέσω των οποίων να επιτρέπεται η επεξεργασία της λίστας των σπουδαστών του «Επίκουρου». Μια τέτοια υλοποίηση, συμπεριλαμβανομένης και της προσθήκης σπουδαστή, είναι η ακόλουθη:

```
public interface StudentsI
{
    public void removeStudent(Student s);
    public void addStudent(Student s);
    public ArrayList difference(ArrayList stlist);
}
```

Δεδομένης της χρήσης της λίστας «ArrayList», η προσθήκη νέου σπουδαστή είναι πολύ εύκολη εργασία. Για να επιτύχουμε αυτό χρειάζεται (και απαιτείται) η κλάση «Students», εφόσον υλοποιεί τη διεπαφή «StudentsI», να υλοποιεί επίσης τη μέθοδο «addStudent». Η υλοποίηση αυτή θα πρέπει να προστεθεί στον κύριο κορμό της κλάσης και έχει ως εξής:

```
public void addStudent(Student s)
{
    studentList.add(s);
}
```

## Άσκηση I/Κεφάλαιο I0

Το συγκεκριμένο ερώτημα είναι αλγοριθμικής φύσης, όμως αποτελεί ένα καλό παράδειγμα προκειμένου να εκτιμήσουμε κάποια από τα προβλήματα που μπορεί να κληθεί να αντιμετωπίσει ένας προγραμματιστής κατά την υλοποίηση.

Όπως έχουμε δει, η συγκεκριμένη υλοποίηση χρησιμοποιεί μια απλή λίστα φοιτητών, στην οποία οι φοιτητές καταχωρούνται με τυχαία σειρά. Αυτό έχει ως αποτέλεσμα η αναζήτηση κάποιου φοιτητή (μέθοδος «findPositionOfStudent» παραπάνω) να μην είναι όσο «καλή» θα μπορούσε να είναι από πλευράς επιδόσεων. Μελετώντας τον κώδικα αυτής της μεθόδου, παρατηρούμε πως ο χρόνος εκτέλεσης της οποιασδήποτε αναζήτησης θα είναι ανάλογος του αριθμού των φοιτητών  $[O(n)]$ . Αν επιλέξουμε μία εναλλακτική υλοποίηση της λίστας φοιτητών χρησιμοποιώντας, για

παράδειγμα, μια λίστα ταξινομημένη βάσει του επωνύμου, θα μπορούσαμε να εφαρμόσουμε αλγόριθμους ταχύτερης αναζήτησης των φοιτητών, όπως ο αλγόριθμος δυαδικής αναζήτησης κ.ά.

Κάτι τέτοιο θα αναμέναμε να πραγματοποιηθεί εξάλλου σε οποιαδήποτε περίπτωση επαγγελματικής ανάπτυξης μιας τέτοιας εφαρμογής. Δηλαδή, μολονότι και οι δύο υλοποιήσεις πληρούν τις λειτουργικές προδιαγραφές της εφαρμογής, δεν είναι εξίσου αποδεκτές για την επιτυχημένη υλοποίηση και λειτουργία της εφαρμογής.

## ΒΙΒΛΙΟΓΡΑΦΙΑ

Khoshafian S., Abnous R., *Object Orientation: Concepts, Languages, Databases, User Interfaces*, Wiley.

Pressman, R. S., *Software Engineering-A Practitioners Approach*, McGraw-Hill.

Sommerville, I. *Software Engineering*, London: Addison-Wesley.



## ΕΛΕΓΧΟΣ ΚΑΙ ΔΙΟΡΘΩΣΗ ΣΦΑΛΜΑΤΩΝ

Σκοπός του κεφαλαίου είναι η ανάδειξη της σημασίας του ελέγχου κατά την ανάπτυξη λογισμικού, καθώς και η παρουσίαση των ενεργειών που γίνονται κατά τη φάση του ελέγχου και των αποτελεσμάτων που παράγονται.

Μετά τη μελέτη του κεφαλαίου, ο αναγνώστης θα είναι σε θέση:

- να υποστηρίξει τη σημασία της τεκμηρίωσης κατά τη φάση της προδιαγραφής των απαιτήσεων και της σχεδίασης λογισμικού,
- να παράγει περιπτώσεις ελέγχου για μονάδες και συστήματα λογισμικού χρησιμοποιώντας τις στρατηγικές που παρουσιάζονται,
- να προγραμματίζει και να εφαρμόζει τα τέσσερα στάδια της διαδικασίας ελέγχου του λογισμικού,
- να περιγράφει τον προγραμματισμό και τα αποτελέσματα του ελέγχου με δομημένα έγγραφα.

### Έννοιες-κλειδιά

---

- Στρατηγική του μαύρου κουτιού
- Στρατηγική του γυάλινου κουτιού
- Δοκιμαστικά δεδομένα
- Πλάνο ελέγχου
- Περίπτωση ελέγχου
- Ισοδύναμη διαμέριση
- Συνοριακές τιμές
- Αναφορές ελέγχου



## Σύνοψη

---

Ο έλεγχος είναι μια ιδιαίτερα σημαντική εργασία στην ανάπτυξη του λογισμικού, η οποία, δυστυχώς, συχνά υποτιμάται, με αποτελέσματα την κακή ποιότητα λογισμικού που συχνά φτάνει στους χρήστες. Η Τεχνολογία Λογισμικού εντάσσει τον έλεγχο μέσα στην ανάπτυξη και προτείνει συγκεκριμένους τρόπους για να προγραμματίζεται και να εκτελείται ο έλεγχος, καθώς και για να περιγράφονται τα αποτελέσματα αυτού. Στο κεφάλαιο αυτό παρουσιάζονται τέσσερις προσεγγίσεις που εντάσσονται στη στρατηγική του μαύρου κουτιού, καθώς και η στρατηγική του γυάλινου κουτιού. Τα σύγχρονα εργαλεία υποστηρίζουν την εκτέλεση πολλών από τις εργασίες αυτές.

## Εισαγωγικές παρατηρήσεις

---

Ένα από τα κυριότερα στάδια κατά την ανάπτυξη ενός λογισμικού συστήματος είναι ο έλεγχος της ορθής λειτουργίας του. Κατά τον έλεγχο θα διαπιστωθεί η καλή λειτουργία του λογισμικού προκειμένου αυτό να δοθεί στους τελικούς χρήστες. Συχνά ο έλεγχος υποτιμάται από τους κατασκευαστές και είτε γίνεται με λάθος τρόπο είτε παραμελείται πλήρως. Τα αποτελέσματα τα βιώνουν οι χρήστες του λογισμικού, οι οποίοι τελικά καλούνται να ανακαλύψουν σφάλματα και παραλείψεις του κατασκευαστή, πράγμα που δεν οφείλουν να κάνουν. Η Τεχνολογία Λογισμικού οφείλει να παρέχει τεχνικές για να σχεδιάζεται, να εκτελείται και να τεκμηριώνεται σωστά ο έλεγχος του λογισμικού αλλά και να αποδίδει σε αυτόν τη σημασία που κατέχει στην ανάπτυξη λογισμικού.

## ΕΝΟΤΗΤΑ 11.1. ΕΛΕΓΧΟΣ ΛΟΓΙΣΜΙΚΟΥ

Ξεκινώντας την ενασχόλησή μας με το θέμα, είναι σκόπιμο να δώσουμε έναν ορισμό της έννοιας του ελέγχου λογισμικού.

### Έλεγχος λογισμικού:

Έλεγχος είναι η διαδικασία κατά την οποία εξετάζεται το λογισμικό με χρήση ειδικά σχεδιασμένων τεχνικών και με σκοπό την εύρεση και διόρθωση σφαλμάτων στην υλοποίησή του.

Τα σφάλματα που αναζητούνται σε αυτό το στάδιο ανάπτυξης κατηγοριοποιούνται σε δύο ομάδες. Στην πρώτη ομάδα διακρίνουμε τα σφάλματα που παρουσιάζονται λόγω αντίφασης των αποτελεσμάτων της λειτουργίας του λογισμικού με συγκεκριμένες απαιτήσεις και προδιαγραφές του συστήματος. Θεωρώντας ότι οι απαιτήσεις από το λογισμικό έχουν οριστεί σωστά, μπορούμε να πούμε ότι στην περίπτωση ύπαρξης των παραπάνω σφαλμάτων το λογισμικό δεν τις ικανοποιεί. Δίνει, δηλαδή, λάθος λύση σε ένα σωστό πρόβλημα. Στην περίπτωση αυτή έχουν γίνει λάθη κατά τη μετάβαση από τις προδιαγραφές στη σχεδίαση του λογισμικού. Ο τρόπος με τον οποίο ελέγχουμε το λογισμικό για τέτοιου είδους λάθη λέγεται «επικύρωση» (validation).

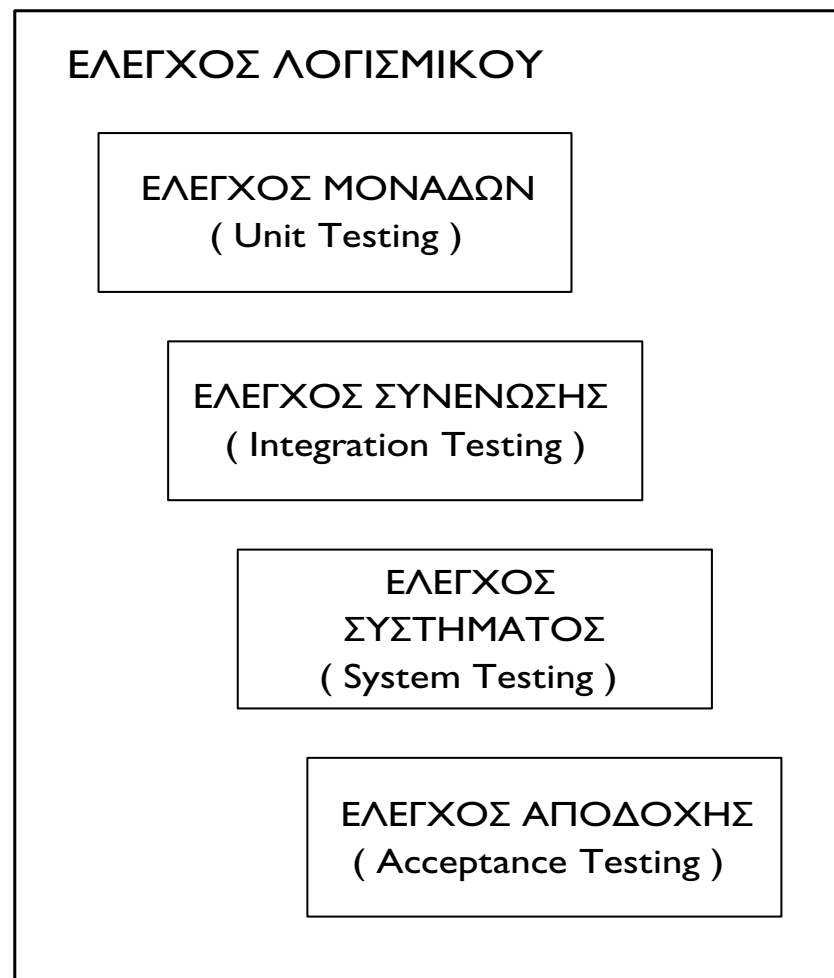
Στη δεύτερη ομάδα διακρίνουμε τα σφάλματα που παρουσιάζονται κατά την εκτέλεση συγκεκριμένων μονάδων του λογισμικού. Στην περίπτωση αυτή η σύλληψη της συγκεκριμένης μονάδας είναι σωστή, αλλά η υλοποίησή της παρουσιάζει λάθη. Ο τρόπος με τον οποίο ελέγχουμε το λογισμικό για τέτοιου είδους λάθη λέγεται «επαλήθευση» (verification).

Έτσι, υπάρχουν δύο διακριτοί τύποι ελέγχου:

- Ο έλεγχος που βασίζεται στις απαιτήσεις από το σύστημα και κατά τον οποίο επαληθεύεται ότι το λογισμικό ανταποκρίνεται σε αυτές.
- Ο έλεγχος κατά τον οποίο επαληθεύεται ότι οι μονάδες του λογισμικού συστήματος έχουν υλοποιηθεί σωστά από προγραμματιστικής άποψης.

Έχοντας κατά νου ότι δεν υπάρχει λογισμικό χωρίς σφάλματα, μπορούμε να πούμε ότι ο έλεγχος κρίνεται επιτυχημένος όταν εντοπίσει σφάλματα και όχι στην αντίθετη περίπτωση. Στην περίπτωση που δεν εντοπιστούν σφάλματα, αυτό δεν σημαίνει και ότι δεν υπάρχουν, αλλά μάλλον ότι ο έλεγχος που επιχειρήθηκε δεν ήταν ικανός να τα αποκαλύψει.

**Σχήμα II.1** Τα επίπεδα εκτέλεσης του ελέγχου.



Ο έλεγχος εκτελείται σε τέσσερα στάδια (Σχήμα II.1), τα οποία αναλύονται στις επόμενες ενότητες, και τεκμηριώνεται με ένα πλάνο ελέγχου το οποίο εξετάζεται στην επόμενη ενότητα.

## ΕΝΟΤΗΤΑ 11.2. ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ ΤΟΥ ΕΛΕΓΧΟΥ

Για τη σωστή εκτέλεση του ελέγχου απαιτείται ο προγραμματισμός του, οι βάσεις για τον οποίο μπορούν να τίθενται οσοδήποτε νωρίς στην ανάπτυξη του λογισμικού. Ο προγραμματισμός αυτός καταγράφεται σε ένα έγγραφο που ονομάζεται «πλάνο ελέγχου» (Test Plan), το οποίο περιέχει πληροφορίες για τον σκοπό του ελέγχου που θα εκτελεστεί, τη στρατηγική που θα χρησιμοποιηθεί και τους αναγκαίους πόρους για τη διεκπεραίωσή του. Ακολουθώντας, στο Σχήμα II.2 παρατίθεται μια δομή του εγγράφου «πλάνο ελέγχου», βασισμένη σε πρότυπο του IEEE (IEEE Standard for Software Test Documentation, ANSI/IEEE, Std 829-1991).

## Σχήμα II.2 Το πλάνο του ελέγχου λογισμικού κατά IEEE.

### Πλάνο ελέγχου

- I. Ταυτότητα του εγγράφου
2. Εισαγωγή
3. Οντότητες που θα ελεγχθούν
4. Χαρακτηριστικά που θα ελεγχθούν
5. Χαρακτηριστικά που δεν θα ελεγχθούν
6. Μέθοδος
7. Κριτήρια επιτυχίας/αποτυχίας ελέγχου οντοτήτων
8. Κριτήρια ακύρωσης και προδιαγραφές επανάληψης ελέγχου.
9. Παραδοτέα έγγραφα
10. Εργασίες που πρέπει να γίνουν
- II. Αναγκαίοι πόροι περιβάλλοντος
12. Κατανομή ευθυνών για την εκτέλεση του ελέγχου
13. Ανάγκες στελέχωσης και εκπαίδευσης προσωπικού
14. Χρονοπρογραμματισμός ελέγχου
15. Κίνδυνοι και απρόοπτα
16. Εγκρίσεις

Το πλάνο ελέγχου δεν είναι απαραίτητο να κατασκευάζεται μετά την παραγωγή του πηγαίου κώδικα. Αντίθετα, όσο νωρίτερα είναι έτοιμο και διαθέσιμο σε όλους τους μετέχοντες στην ανάπτυξη του λογισμικού, τόσο περισσότερο διευκολύνεται η εκτέλεση του ελέγχου. Η διευκόλυνση αυτή μπορεί να εντοπιστεί σε δύο επίπεδα: πρώτον, στην ετοιμασία και διάθεση των αναγκαίων για τον έλεγχο εγγράφων και, δεύτερον, στη δυνατότητα έναρξης της διαδικασίας του ελέγχου προτού ολοκληρωθεί η παραγωγή του πηγαίου κώδικα, πράγμα που είναι και η καλύτερη μέθοδος που μπορεί να ακολουθήσει κανείς.

### **Δραστηριότητα I/Κεφάλαιο II**

Αναφέρετε τους δύο σημαντικότερους λόγους για τους οποίους το πλάνο του ελέγχου μπορεί να ξεκινήσει να γράφεται μετά από την ολοκλήρωση της φάσης των προδιαγραφών των απαιτήσεων από το λογισμικό.

## **ΕΝΟΤΗΤΑ 11.3. ΤΕΧΝΙΚΕΣ ΕΛΕΓΧΟΥ**

Ο έλεγχος μιας εφαρμογής λογισμικού στηρίζεται στην αρχή ότι εκτελείται ένα τμήμα αυτής με ένα σύνολο από δεδομένα εισόδου για τα οποία τα αποτελέσματα είναι γνωστά και, αν τα αποτελέσματα που λαμβάνονται από την εκτέλεση δεν είναι ίδια με τα αναμενόμενα, τότε το τμήμα αυτό έχει σφάλματα.

### **Δοκιμή μονάδας:**

Η εκτέλεση μιας μονάδας προγράμματος με ένα σύνολο από δεδομένα για τα οποία τα αποτελέσματα είναι γνωστά λέγεται «δοκιμή» (test) της μονάδας. Δοκιμές που γίνονται με διαφορετικά δεδομένα θεωρούνται διαφορετικές.

Τα δεδομένα εισόδου και εξόδου που χρησιμοποιούνται κατά τις δοκιμές λέγονται «δοκιμαστικά δεδομένα» (test data). Ακολουθώς εισάγεται και ο όρος της περίπτωσης ελέγχου.

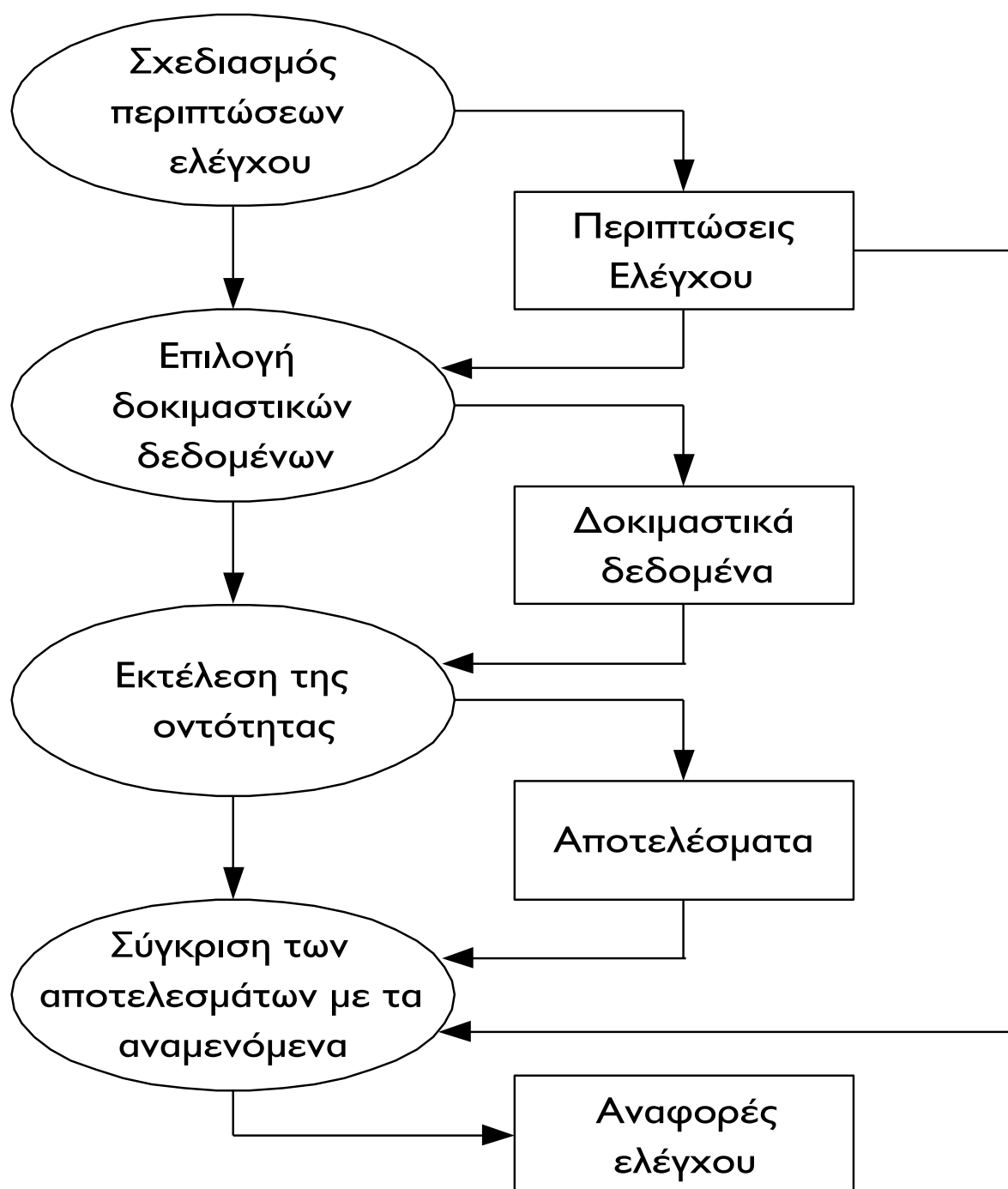
### **Περίπτωση ελέγχου:**

Μια περίπτωση ελέγχου (test case) είναι το σύνολο των δοκιμαστικών δεδομένων, των συνθηκών εκτέλεσης και των αναμενόμενων αποτελεσμάτων που έχουν σχεδιαστεί με έναν συγκεκριμένο σκοπό, όπως το να καλύψουν ένα μονοπάτι εκτέλεσης του λογισμικού ή να επικυρώσουν τη συμφωνία με μια συγκεκριμένη απαίτηση από αυτό.

Έχοντας κατά νου τους παραπάνω ορισμούς, στο Σχήμα II.3 φαίνεται η γενική ροή ελέγχου μιας μονάδας λογισμικού. Αρχικά, σχεδιάζονται οι περιπτώσεις ελέγχου, δηλαδή ορίζονται τα δοκιμαστικά δεδομένα, οι συνθήκες εκτέλεσης και τα αναμενόμενα αποτελέσματα για κάθε δοκιμή. Στη συνέχεια, εκτελείται το ελεγχόμενο λογισμικό με τα δοκιμαστικά δεδομένα κάθε περίπτωσης ελέγχου και λαμβάνονται κάποια αποτελέσματα τα οποία καταγράφονται και συγκρίνονται με τα αναμενόμενα αποτελέσματα των αντίστοιχων περιπτώσεων ελέγχου ώστε να συνταχτούν οι απαραίτητες αναφορές ελέγχου.



**Σχήμα II.3** Γενική ροή ελέγχου μιας μονάδας λογισμικού.

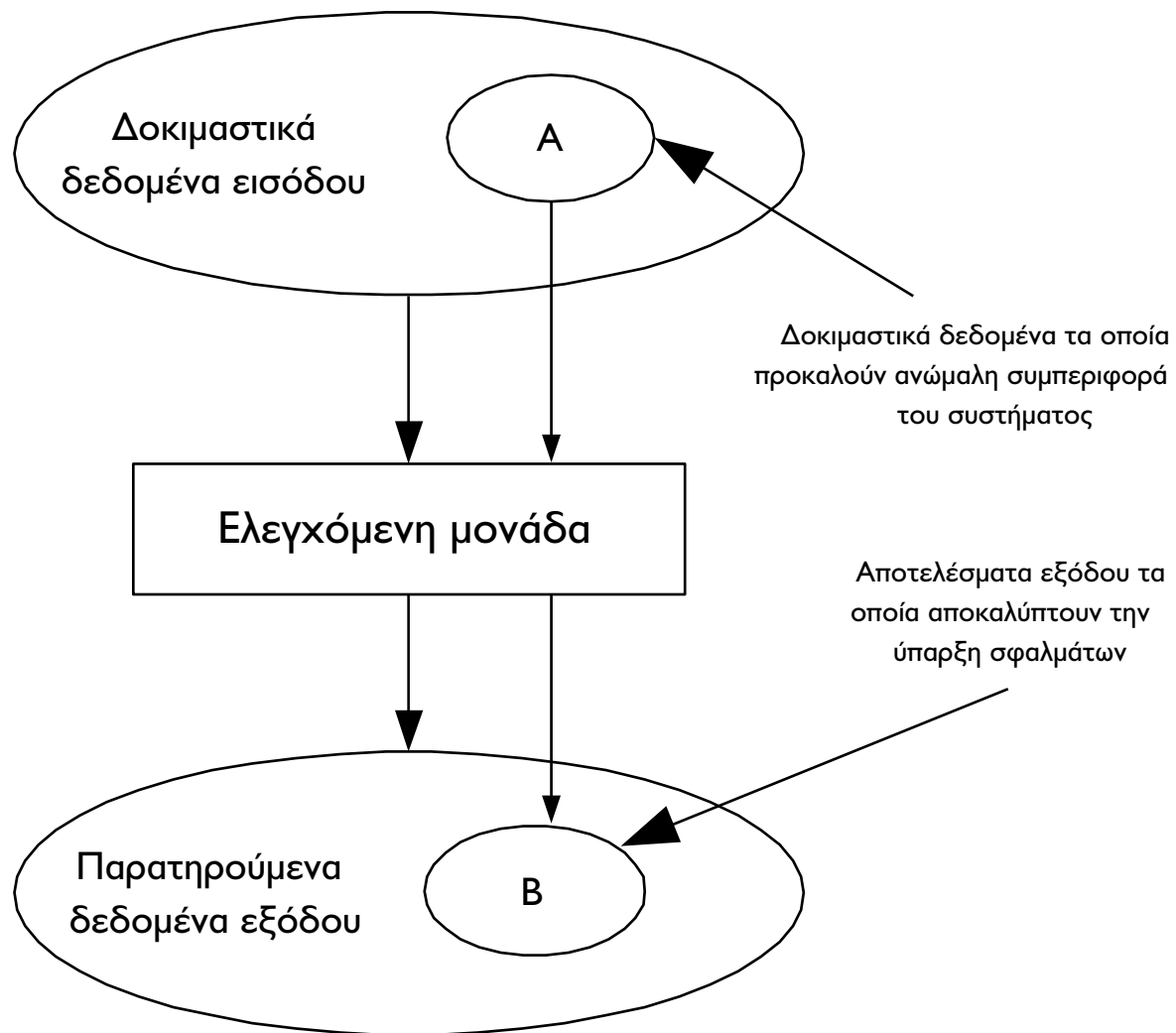


Δεν είναι εφικτό να δοκιμαστούν όλες οι δυνατές περιπτώσεις ελέγχου καθώς ο αριθμός τους είναι απαγορευτικά μεγάλος. Επομένως, θα πρέπει να γίνεται μια επιλογή ορισμένων μόνο από αυτές, η οποία θα πρέπει να είναι τέτοια ώστε να ικανοποιείται ο σκοπός του ελέγχου. Υπάρχουν δύο βασικές στρατηγικές που ακολουθούνται στην πράξη για τη λύση του προβλήματος αυτού, οι οποίες διακρίνονται ως προς την άποψη από την οποία παρατηρούν το λογισμικό: η στρατηγική του μαύρου κουτιού και η στρατηγική του γυάλινου κουτιού.

### **II.3.1. Στρατηγική του μαύρου κουτιού.**

Η στρατηγική του μαύρου κουτιού (black-box testing) βασίζεται στις προδιαγραφές της μονάδας λογισμικού που πρόκειται να ελεγχθεί, οι οποίες θεωρούνται γνωστές, ενώ ο τρόπος κατασκευής της θεωρείται άγνωστος. Αντιμετωπίζεται δηλαδή σαν ένα μαύρο κουτί του οποίου η συμπεριφορά μπορεί να μελετηθεί μόνο παρατηρώντας τις εισόδους και τα αποτελέσματα που αντιστοιχούν σε αυτές (Σχήμα II.4).

**Σχήμα II.4** Έλεγχος λογισμικού με τη στρατηγική του μαύρου κουτιού.



Για την αποκάλυψη όλων των σφαλμάτων αρκεί να δοκιμαστούν τα δεδομένα που ανήκουν στο σύνολο  $A$ , όπως φαίνεται στο παραπάνω σχήμα, δηλαδή τα δοκιμαστικά δεδομένα που προκαλούν ανώμαλη συμπεριφορά του συστήματος. Στην ουσία όμως, για να γίνει κάτι τέτοιο θα πρέπει να δοκιμαστούν όλα τα δυνατά δεδομένα εισόδου, κάτι το οποίο είναι ανέφικτο. Θα πρέπει, λοιπόν, να γίνει επιλογή ενός συνόλου περιπτώσεων ελέγχου οι οποίες έχουν μεγάλη πιθανότητα να αποκαλύψουν σφάλματα στην ελεγχόμενη οντότητα. Για τον σκοπό αυτό έχουν αναπτυχθεί ορισμένες προσεγγίσεις, οι οποίες αναλύονται στη συνέχεια.

## Προσέγγιση της ισοδύναμης διαμέρισης

Τα δεδομένα εισόδου σε ένα λογισμικό σύστημα μπορούν συνήθως να κατηγοριοποιηθούν σε έναν αριθμό διαφορετικών κλάσεων. Οι κλάσεις αυτές έχουν κοινά χαρακτηριστικά (π.χ. θετικοί αριθμοί, αρνητικοί αριθμοί, συμβολοσειρές χωρίς κενά κ.λπ.). Μια μονάδα λογισμικού συνήθως συμπεριφέρεται με παρόμοιο τρόπο για όλα τα μέλη καθεμίας από αυτές τις κλάσεις. Δηλαδή, αν μια τιμή αποκαλύψει ένα λάθος κατά τον έλεγχο, τότε το ίδιο λάθος θα αποκαλύψει και μια οποιαδήποτε από τις υπόλοιπες τιμές της ίδιας κλάσης. Επίσης, αν μια τιμή δεν αποκαλύψει κάποιο λάθος, το ίδιο θα κάνει και μία από τις υπόλοιπες τιμές. Εξαιτίας αυτής της «ισοδύναμης» συμπεριφοράς, οι κλάσεις αυτές ονομάζονται «κλάσεις ισοδύναμων τιμών» (equivalence partitions).

### Κλάση ισοδύναμων τιμών:

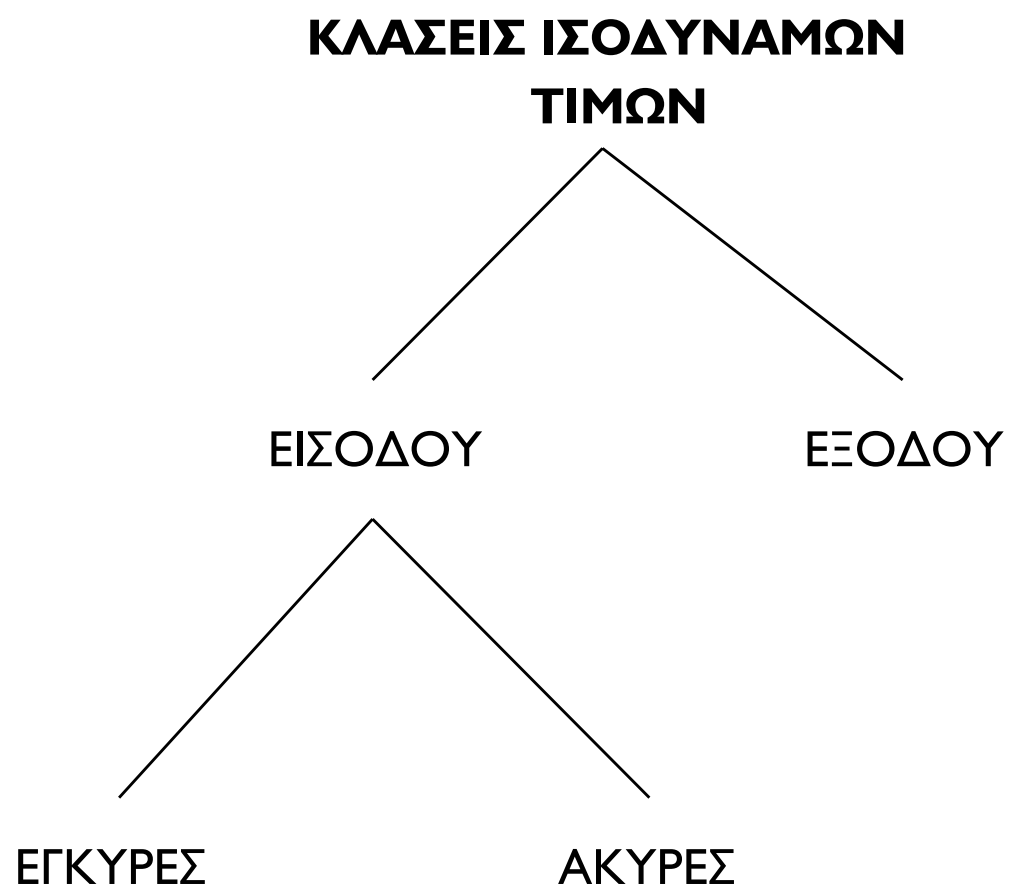
Μια κλάση ισοδύναμων τιμών είναι ένα σύνολο με μέλη είτε δοκιμαστικά δεδομένα εισόδου είτε δεδομένα εξόδου τα οποία προσδιορίζουν μια ισοδύναμη συμπεριφορά του συστήματος.

Η προσέγγιση της ισοδύναμης διαμέρισης βασίζεται στον προσδιορισμό ενός συνόλου κλάσεων ισοδύναμων τιμών. Οι περιπτώσεις ελέγχου στη συνέχεια θα πρέπει να σχεδιάζονται έτσι ώστε τα δοκιμαστικά δεδομένα αλλά και τα αναμενόμενα αποτελέσματα που τις χαρακτηρίζουν να ανήκουν σε κάποια από αυτές τις κλάσεις. Με τον τρόπο αυτό μειώνεται το πλήθος των

περιπτώσεων ελέγχου, καθώς πλέον το πολύ να είναι ίσο με το πλήθος των κλάσεων των ισοδύναμων τιμών, αφού πια δεν χρειάζεται να ελέγξουμε παραπάνω από ένα δοκιμαστικό δεδομένο μιας κλάσης καθώς θα πάρουμε το ίδιο αποτέλεσμα ως προς την ορθότητα.

Οι κλάσεις ισοδύναμων τιμών μπορούν να διακριθούν σε κλάσεις ισοδύναμων τιμών εισόδου και σε κλάσεις ισοδύναμων τιμών εξόδου. Οι πρώτες έχουν μέλη δοκιμαστικά δεδομένα, ενώ οι δεύτερες έχουν μέλη δεδομένα εξόδου (αποτελέσματα εκτέλεσης). Επίσης, είναι δυνατή η διάκριση των κλάσεων ισοδύναμων τιμών εισόδου σε έγκυρες και άκυρες, ανάλογα με το αν προκαλούν ομαλή ή όχι λειτουργία της μονάδας που ελέγχεται (Σχήμα II.5).

**Σχήμα II.5** Διάκριση των κλάσεων ισοδυνάμων τιμών.



Ο προσδιορισμός των κλάσεων ισοδύναμων τιμών εισόδου βασίζεται στους περιορισμούς που υπάρχουν στα δεδομένα εισόδου (συνθήκες εισόδου). Από τις συνθήκες εισόδου μπορούν να παραχθούν κλάσεις ισοδύναμων τιμών με έναν ευριστικό (δηλαδή αυθαίρετο και πρακτικά σωστό) τρόπο. Μερικές κατευθυντήριες γραμμές για τον προσδιορισμό των κλάσεων αυτών είναι οι εξής:

- Αν μια συνθήκη εισόδου προδιαγράφει ένα διάστημα τιμών, τότε υπάρχει μια έγκυρη κλάση ισοδύναμων τιμών (το δεδομένο εισόδου ανήκει στο συγκεκριμένο διάστημα) και μία ή δύο άκυρες κλάσεις (το δεδομένο εισόδου δεν ανήκει στο συγκεκριμένο διάστημα).
- Αν μια συνθήκη εισόδου προδιαγράφει πλήθος αλλά και τη μέγιστη τιμή του, τότε υπάρχει μία έγκυρη κλάση ισοδύναμων τιμών (το δεδομένο εισόδου να είναι μεγαλύτερο του μηδενός και το πολύ ίσο με τη μέγιστη τιμή του) και δύο άκυρες κλάσεις (το δεδομένο εισόδου δεν ανήκει στο προηγούμενο διάστημα).
- Αν μια συνθήκη προδιαγράφει ένα σύνολο από τιμές για τις οποίες υπάρχει η υπόνοια ότι η προς έλεγχο οντότητα τις μεταχειρίζεται διαφορετικά, τότε υπάρχουν τόσες έγκυρες κλάσεις ισοδύναμων τιμών όσες είναι οι τιμές του συνόλου και μία άκυρη κλάση με μία μόνο τιμή τέτοια που να είναι διαφορετική από τις τιμές του συνόλου αυτού.

Αφού, λοιπόν, προσδιοριστούν οι κλάσεις ισοδύναμων τιμών, το επόμενο βήμα είναι η επιλογή των περιπτώσεων ελέγχου, η οποία μπορεί να γίνει σύμφωνα με τα παρακάτω βήματα:

- Μέχρι να καλυφθούν όλες οι έγκυρες κλάσεις από περιπτώσεις ελέγχου, ορίζεται μία καινούρια περίπτωση ελέγχου που να καλύπτει όσο γίνεται περισσότερες από τις ακάλυπτες έγκυρες κλάσεις.
- Μέχρι να καλυφθούν οι άκυρες κλάσεις από περιπτώσεις ελέγχου, ορίζεται μία καινούρια περίπτωση ελέγχου που να καλύπτει μία και μόνο μία από τις ακάλυπτες άκυρες κλάσεις.

## Προσέγγιση συνοριακών τιμών

Συνοριακές τιμές (boundary values) σε μια κλάση ισοδύναμων τιμών είναι οι τιμές των άκρων της. Έχει παρατηρηθεί ότι κατά την ανάπτυξη μιας εφαρμογής λογισμικού οι σχεδιαστές και οι προγραμματιστές τείνουν να λαμβάνουν υπόψη τους τυπικές τιμές για τα δεδομένα εισόδου, ενώ συνήθως παραβλέπουν τις συνοριακές τιμές. Η αντιμετώπιση αυτή συμβάλλει στη γέννηση πολλών σφαλμάτων πάνω στις συνοριακές τιμές, καθώς και σε τιμές πριν ή μετά από αυτές. Είναι, λοιπόν, μια καλή στρατηγική ελέγχου να ελέγχονται μεθοδικά οι τιμές αυτές. Κάτι τέτοιο ισχύει όχι μόνο για τις κλάσεις ισοδύναμων τιμών εισόδου αλλά και για τις κλάσεις ισοδύναμων τιμών εξόδου. Συμπληρώνουμε, λοιπόν, τις περιπτώσεις ελέγχου που παράγονται με την προσέγγιση της ισοδύναμης διαμέρισης σύμφωνα με τις ακόλουθες κατευθυντήριες γραμμές:

- Αν μια συνθήκη εισόδου προδιαγράφει ένα διάστημα τιμών, ορίζονται έγκυρες περιπτώσεις ελέγχου για τα άκρα του διαστήματος και άκυρες για τις τιμές ακριβώς έξω από τα άκρα.
- Αν μια συνθήκη εισόδου προδιαγράφει ένα πλήθος από τιμές, ορίζονται περιπτώσεις ελέγχου με το μικρότερο και το μεγαλύτερο πλήθος τιμών, καθώς και με ένα μικρότερο ή ένα μεγαλύτερο αντίστοιχα.
- Αντίστοιχα ορίζονται περιπτώσεις ελέγχου για κάθε συνθήκη εξόδου.
- Αν η είσοδος ή η έξοδος είναι ένα διατεταγμένο σύνολο (π.χ. ένα ακολουθιακό αρχείο, μια γραμμική λίστα), ορίζονται περιπτώσεις ελέγχου για το πρώτο και το τελευταίο στοιχείο του συνόλου.

## Προσέγγιση αιτίου – αποτελέσματος

Η επιλογή των περιπτώσεων ελέγχου στην προσέγγιση αιτίου – αποτελέσματος (cause and effect graphing) γίνεται με τη βοήθεια ενός γράφου. Ο γράφος αυτός στους αρχικούς του κόμβους έχει αίτια, στους ενδιάμεσους έχει περιορισμούς και στους τελικούς έχει αποτελέσματα. Με τη βοήθεια



του κόμβου αυτού κατασκευάζεται ένας πίνακας απόφασης (decision table) από τον οποίο μπορούν μηχανιστικά να παραχθούν περιπτώσεις ελέγχου. Το πλεονέκτημα της προσέγγισης αυτής είναι ότι θεωρεί συνδυασμούς συνθηκών εισόδου, και οι παραγόμενες περιπτώσεις ελέγχου είναι πολύ αποτελεσματικές. Στην πραγματικότητα όμως ο γράφος γίνεται πολύ μεγάλος και άβολος, με αποτέλεσμα η όλη διεργασία εξαγωγής περιπτώσεων ελέγχου να είναι πολύ κοπιαστική.

## **Προσέγγιση μαντέματος**

Υπάρχουν πολλοί άνθρωποι οι οποίοι με τη διορατικότητα που τους διακρίνει επιλέγουν περιπτώσεις ελέγχου ικανές να αποκαλύπτουν σφάλματα στο λογισμικό. Σε αυτό ακριβώς το γεγονός βασίζεται η προσέγγιση του μαντέματος. Βέβαια, στις περισσότερες περιπτώσεις η διορατικότητα που διακρίνει ένα άτομο είναι απόρροια των γνώσεων και της πείρας που διαθέτει. Αν και αυτή η μεθοδολογία είναι κάπως ανεπίσημη, δεν μπορούμε να την παραβλέψουμε, καθώς συχνά είναι ιδιαίτερα αποτελεσματική.

## Δραστηριότητα 2/Κεφάλαιο II

Ας θεωρήσουμε την ακόλουθη μονάδα προγράμματος, γραμμένη σε γλώσσα Pascal, η οποία διαβάζει μια ημερομηνία από το πληκτρολόγιο και την επιστρέφει σαν παράμετρο στη μονάδα που την καλεί.

```
Procedure GetDate(var d,m,y: integer; var flag: integer);  
Var day, month, year: integer;  
1. Begin  
2.   day:=1; month:=1; year:=1900; flag:=0;  
3.   Write('Δώσε ημερομηνία (η/μ/ε):');  
4.   Read(day); Read(month); Readln(year);  
5.   If (month<1) or (month>12) then  
6.     Flag:=1;  
7.   If (day<1) or (day>31) then  
8.     Flag:=1;  
9.   If Flag=0 then  
10.    Begin  
11.      If month in [1,3,5,7,8,10,12] then  
12.        UpperDayValue:=31;  
13.      If month in [4,6,9,11] then  
14.        UpperDayValue:=30;  
15.      If month=2 then  
16.        If leap(year) then  
17.          UpperDayValue:=29  
18.        Else  
19.          UpperDayValue:=28;  
20.      If day>UpperDayValue Then  
21.        Flag:=1  
22.      Else  
23.        Begin  
24.          D:=day;  
25.          M:=month;  
26.          Y:=year  
27.        End;  
28.    End;  
29. End;
```

Στο παραπάνω πρόγραμμα ο αναγνώστης καλείται να κατασκευάσει περιπτώσεις ελέγχου σύμφωνα με τις προσεγγίσεις της ισοδύναμης διαμέρισης

και των συνοριακών τιμών. Μεταβλητές εισόδου είναι οι ακέραιες μεταβλητές  $d, m, y$ .

Επεξήγηση του προγράμματος: Αρχικά γίνονται οι δηλώσεις του ονόματος της διαδικασίας και των παραμέτρων (αυτές που θα επιστραφούν στην καλούσα μονάδα έχουν τον χαρακτηρισμό «var»), καθώς και των μεταβλητών που θα χρησιμοποιηθούν εσωτερικά στην «GetDate». Το πρόγραμμα δίνει αρχικές τιμές στις μεταβλητές (γραμμή 2), τυπώνει ένα μήνυμα (γραμμή 3) και περιμένει να διαβάσει από το πληκτρολόγιο τις τιμές τριών μεταβλητών που αντιστοιχούν στην ημερομηνία που εισάγει ο χρήστης (γραμμή 4). Ακολούθως ελέγχει τα όρια του μήνα (γραμμή 5-6) και της ημερομηνίας (γραμμή 7-8) και, αν είναι εκτός των επιτρεπτών τιμών, δίνει τιμή στη μεταβλητή *flag*, η οποία περιέχει την πληροφορία ύπαρξης σφάλματος (αν έχει τιμή διάφορη του μηδενός) ή όχι (αν έχει τιμή ίση με το μηδέν). Μέχρι το σημείο αυτό έχει εξασφαλιστεί ότι έχει δοθεί τιμή για τον μήνα μεταξύ 1 και 12 και τιμή για την ημερομηνία μεταξύ 1 και 31. Αυτό δεν αποκλείει να έχει δοθεί μη έγκυρη ημερομηνία, για παράδειγμα. 31.2.2000 ή 31.4.1999. Ο έλεγχος για αυτό θα γίνει ακολούθως.

Το τμήμα από τη γραμμή 9 μέχρι και την 28 εκτελείται μόνο αν δεν υπάρχει (μέχρι το σημείο εκείνο) σφάλμα. Στις γραμμές 11 έως 19 καθορίζεται η μέγιστη επιτρεπόμενη τιμή της ημερομηνίας ανάλογα με την τιμή του μήνα. Θεωρείται ότι υπάρχει η συνάρτηση «*leap*», η οποία επιστρέφει τη λογική τιμή «αληθές» αν ο χρόνος που δίνεται ως όρισμά της είναι δίσεκτος, διαφορετικά επιστρέφει την τιμή «ψευδές».

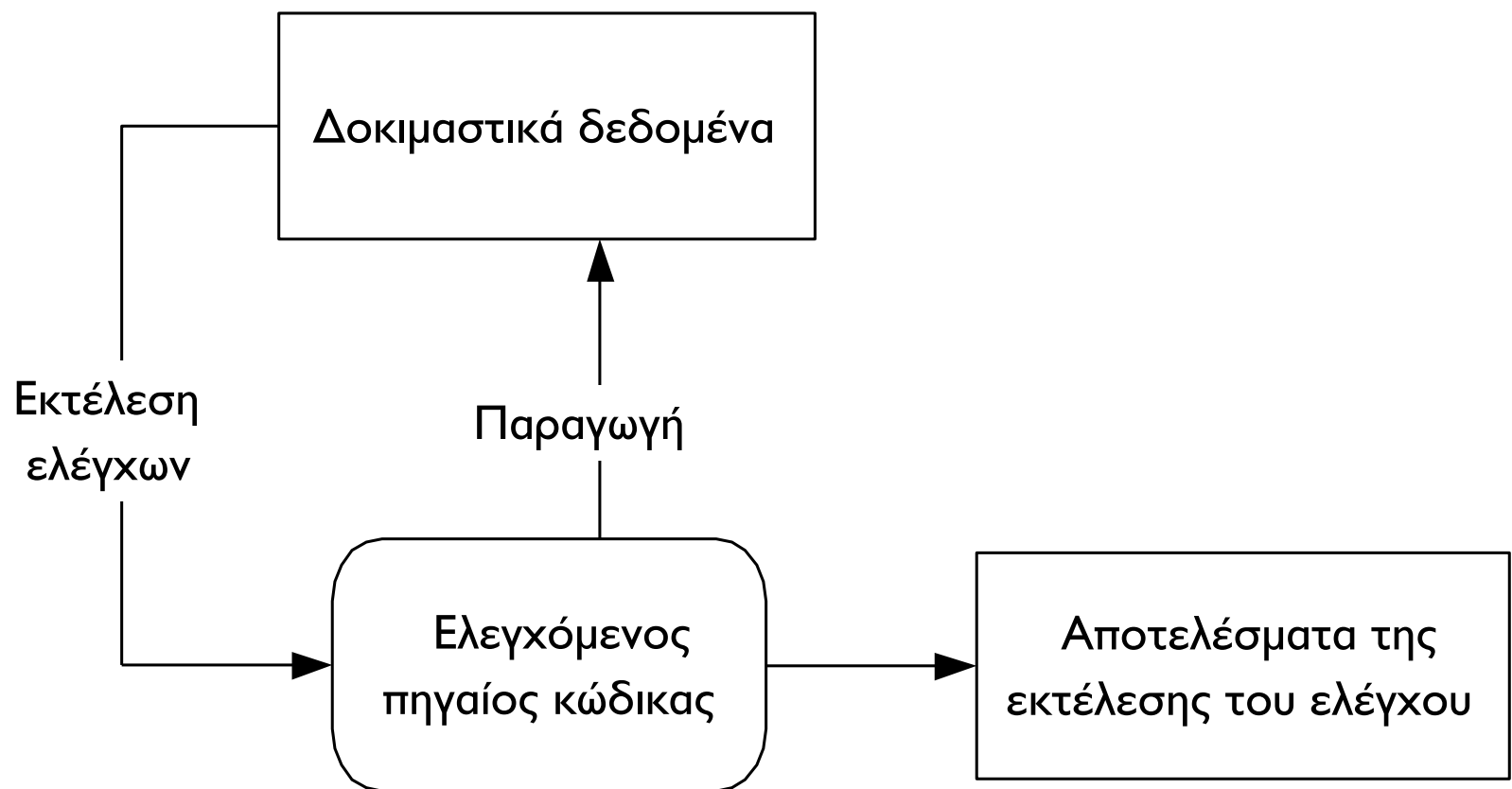
Ο μοναδικός έλεγχος που μένει είναι να διαπιστωθεί αν η ημερομηνία που δόθηκε είναι εντός του επιτρεπτού ορίου, το οποίο έχει καθοριστεί ανάλογα με τον μήνα και το δίσεκτο ή όχι έτος. Αυτός πραγματοποιείται στις γραμμές 20 και 21 και, αν είναι επιτυχής, δίνονται οι τιμές στις παραμέτρους εξόδου της διαδικασίας.

### II.3.2. Στρατηγική του γυάλινου κουτιού

Στη στρατηγική αυτή το λογισμικό που ελέγχεται αντιμετωπίζεται ως ένα διαφανές ή γυάλινο κουτί (*glass box*). Αυτό γιατί θεωρείται γνωστή (δηλαδή

είναι ορατή) η δομή και ο τρόπος λειτουργίας του, οπότε δεν αποτελεί μαύρο κουτί, όπως στην προηγούμενη περίπτωση. Η επιλογή των περιπτώσεων ελέγχου γίνεται μετά από μελέτη του κώδικα και της δομής της υπό έλεγχο οντότητας. Απαραίτητα για την εργασία αυτή είναι το λεπτομερές σχέδιο της υπό έλεγχο μονάδας λογισμικού, καθώς και ο πηγαίος της κώδικας. Στο Σχήμα II.6 δίνεται η ροή εργασιών κατά την εφαρμογή της στρατηγικής του γυάλινου κουτιού.

**Σχήμα II.6** Στρατηγική του γυάλινου κουτιού.



Για την επιλογή περιπτώσεων ελέγχου στη στρατηγική αυτή χρησιμοποιούνται διάφορα κριτήρια, τα οποία λέγονται «κριτήρια κάλυψης». Βέβαια, είναι αυτονόητο ότι όσο περισσότερες περιπτώσεις ελέγχου επιλέγονται για το κάθε κριτήριο, με τόσο περισσότερη ασφάλεια μπορεί να εγγυηθεί κάποιος ότι το έχει καλύψει. Μερικά χρήσιμα κριτήρια κάλυψης είναι τα ακόλουθα.

- Όλα τα ανεξάρτητα μονοπάτια εκτέλεσης του ελεγχόμενου κώδικα πρέπει να εκτελούνται τουλάχιστον μία φορά, δηλαδή δεν πρέπει να υπάρχουν γραμμές πηγαίου κώδικα των οποίων η εκτέλεση να μην ελέγχεται.
- Όλες οι διακλαδώσεις ροής με εντολές τύπου «if - then – else» πρέπει να εκτελούνται τουλάχιστον μία φορά, τόσο για την περίπτωση αληθούς συνθήκης όσο και ψευδούς.
- Όλοι οι βρόγχοι επανάληψης θα πρέπει να εκτελούνται και για ορισμένες τιμές επαναλήψεων αλλά και για αριθμό επαναλήψεων εντός των ορίων. Για παράδειγμα, ένας βρόγχος με μέγιστο αριθμό επαναλήψεων  $n$  θα πρέπει να εκτελεστεί για  $0, 1, 2, \mu$  (όπου  $\mu < n$ ),  $n-1, n$  και  $n+1$  επαναλήψεις.

Για να αποκαλυφθούν όλα τα σφάλματα της ελεγχόμενης οντότητας, πρέπει να δοκιμαστούν όλα τα δυνατά μονοπάτια εκτέλεσής της και όχι απλά να εκτελεστεί μία φορά κάθε γραμμή προγράμματος. Εκτός τετριμμένων περιπτώσεων, κάτι τέτοιο είναι ανέφικτο, όπως συμβαίνει και με τη στρατηγική του μαύρου κουτιού. Γενικά, η εμπειρία δείχνει ότι ο εξαντλητικός έλεγχος του μαύρου κουτιού είναι καλύτερος από αυτόν του γυάλινου κουτιού.

## Παράδειγμα I/Κεφάλαιο II

Ακολουθεί ένα παράδειγμα εκτέλεσης ελέγχου μιας μονάδας προγράμματος με τη στρατηγική του άσπρου κουτιού. Η εν λόγω μονάδα λύνει τη δευτεροβάθμια εξίσωση  $a x^2 + b x + c = 0$ . Συγκεκριμένα, δέχεται ως είσοδο τις παραμέτρους  $a, b, c$  και επιστρέφει τη λύση  $x$  (μεταβλητές  $x^1$  και  $x^2$ ) της εξίσωσης (αν έχει) και μία παράμετρο  $flag$  που δηλώνει αν η εξίσωση έχει δυο λύσεις (η  $flag$  θα πάρει την τιμή 0), διπλή λύση (τιμή 1) ή είναι αδύνατη στο σύνολο των πραγματικών αριθμών (τιμή 2). Ο κώδικας της μονάδας, γραμμένος σε γλώσσα Pascal, φαίνεται παρακάτω :

```

Procedure equ (a,b,c:real; var x1,x2:real; var flag:integer) ;
Var d:real;
1.   Begin
2.       d=b*b-4*a*c;
3.       if (d<0) or (a=0) then flag:=2
4.       else if d>0 then flag:=0
5.       else flag:=1;
6.       if flag=0 then
7.           begin
8.               x1:=(-b+sqrt(d))/2*a;
9.               x2:=(-b-sqrt(d))/2*a;
10.          writeln('Η εξίσωση έχει δύο διαφορετικές λύσεις')
11.      end
12.      else if flag=1 then
13.          begin
14.              x1:=(-b)/2*a;
15.              x2:=x1;
16.          writeln('Η εξίσωση έχει μια διπλή λύση')
17.      end
18.      else writeln('Η εξίσωση δεν έχει λύση στο R')
19.  end;

```

Στην παραπάνω μονάδα προγράμματος οι μεταβλητές εισόδου είναι οι  $a$ ,  $b$  και  $c$ . Έτσι, για κάθε περίπτωση ελέγχου θα πρέπει να επιλέξουμε μια τριάδα τιμών για τις μεταβλητές αυτές οι οποίες και θα αποτελούν τα δοκιμαστικά δεδομένα εισόδου. Οι επιλογές αυτές θα πρέπει να είναι τέτοιες, ώστε με το πέρας όλων των ελέγχων να έχουν εκτελεστεί τουλάχιστον μία φορά όλες οι εντολές της μονάδας, καθώς και όλες οι εντολές απόφασης τόσο στην περίπτωση που οι συνθήκες έχουν τιμή αλήθειας όσο και στην περίπτωση που έχουν τιμή ψεύδους.

Για την επίδειξη της εφαρμογής της στρατηγικής του άσπρου κουτιού δίνεται στη συνέχεια ένας πίνακας, η κάθε γραμμή του οποίου αντιστοιχεί στην



εκτέλεση μιας περίπτωσης ελέγχου. Για κάθε τέτοια περίπτωση δίνεται το μονοπάτι εκτέλεσης εντολών (καταγράφονται οι γραμμές προγράμματος που εκτελούνται), καθώς και η κάλυψη συνθηκών, δηλαδή οι εντολές απόφασης που εκτελούνται και η τιμή της συνθήκης για καθεμία από αυτές.

Δοκιμαστικά Δεδομένα			Κάλυψη εντολών (μονοπάτι εκτέλεσης)	Κάλυψη συνθηκών (Εντολές απόφασης – τιμές συνθηκών)
A	B	C		
0	1	2	1-2-3-6-12-18-19	3: F-T, 6: F, 12: F, 18: T
1	3	1	1-2-3-4-6-7-8-9-10-11-19	3: F-F, 4: T, 6: T
4	-4	1	1-2-3-4-5-6-12-13-14-15-16-17-19	3: F-F, 4: F, 5: T, 6: F, 12: T

Όπως φαίνεται στον παραπάνω πίνακα, οι εντολές της μονάδας που ελέγχεται εκτελούνται τουλάχιστον για μία φορά. Αντίθετα, οι συνθήκες σε ορισμένες από τις εντολές απόφασης δεν παίρνουν και τις δύο δυνατές τιμές τους. Μία ή και περισσότερες περιπτώσεις ελέγχου είναι απαραίτητες για να καλυφθεί και αυτό το κριτήριο, και η επιλογή τους αφήνεται ως άσκηση για τον αναγνώστη.

### Δραστηριότητα 3/Κεφάλαιο II

Στο παραπάνω παράδειγμα συμπληρώστε τις περιπτώσεις ελέγχου, ώστε να ελέγχονται οι συνδυασμοί τιμών στις εντολές απόφασης.

### Άσκηση I/Κεφάλαιο II

Στο παράδειγμα της μονάδας προγράμματος που διαβάζει την ημερομηνία (Ενότητα II.3.I), δώστε περιπτώσεις ελέγχου, έτσι ώστε να εξασφαλίζεται η πλήρης κάλυψη του πηγαίου κώδικα της μονάδας.

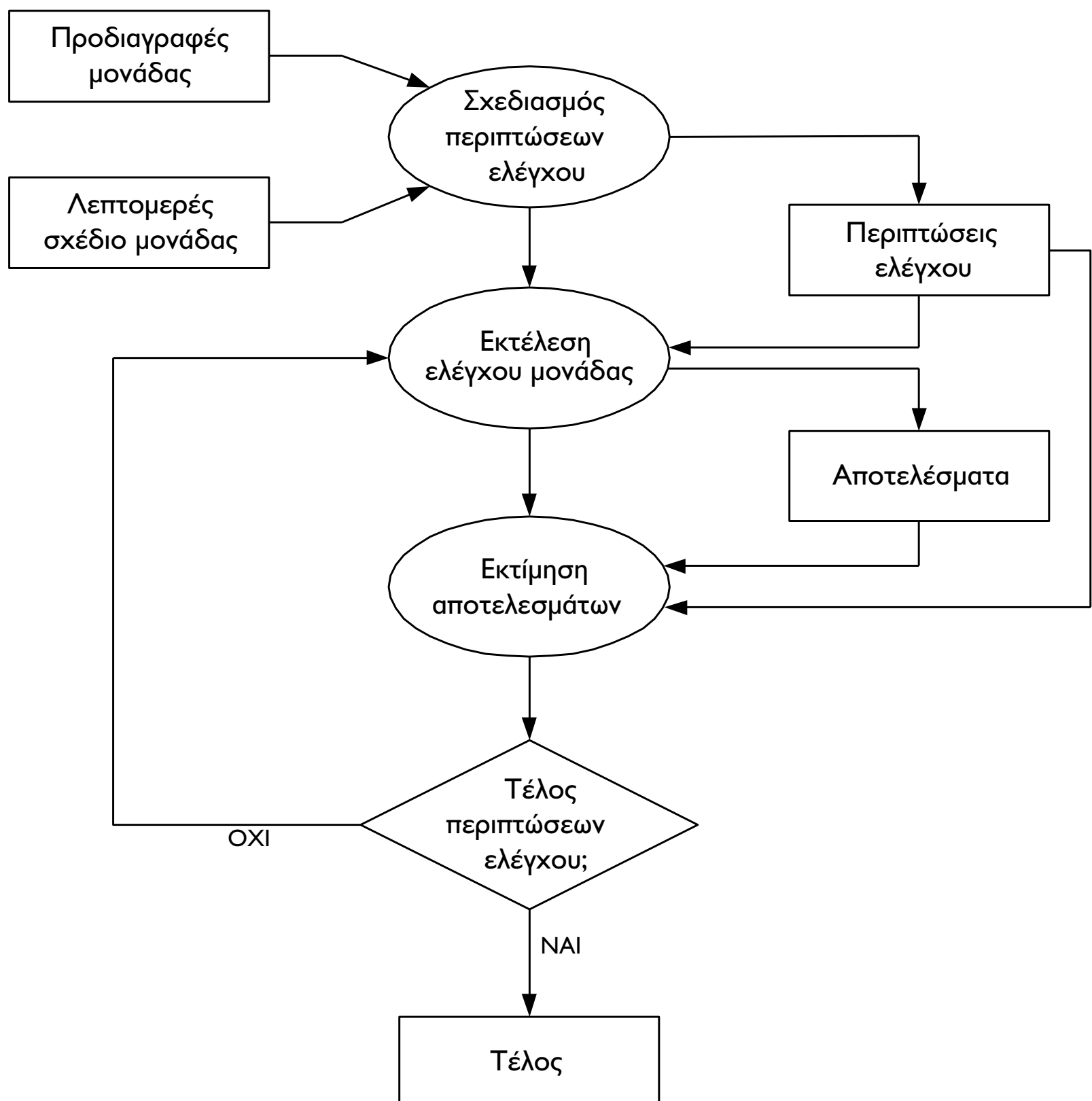
## ΕΝΟΤΗΤΑ 11.4. ΕΚΤΕΛΕΣΗ ΕΛΕΓΧΟΥ

Ο έλεγχος μιας εφαρμογής λογισμικού πραγματοποιείται σε τέσσερα στάδια. Τα στάδια αυτά, από το κατώτερο επίπεδο στο ανώτερο, είναι ο έλεγχος μονάδας (unit testing), ο έλεγχος συνένωσης (integration testing), ο έλεγχος συστήματος (system testing) και ο έλεγχος αποδοχής (acceptance testing). Για τη συστηματική εκτέλεση του ελέγχου προχωρούμε πάντα από το κατώτερο επίπεδο στο ανώτερο, καθώς η λογική που υιοθετείται είναι να ελέγχεται πρώτα το μέρος και μετά το όλον. Εξάλλου, αυτή είναι και η σειρά με την οποία κατασκευάζονται τα συστατικά του λογισμικού. Στο πρώτο επίπεδο, ελέγχεται ο κώδικας μιας μονάδας του συστήματος, στο δεύτερο ο κώδικας περισσότερων της μίας μονάδας, στο τρίτο και τέταρτο ο κώδικας όλου του λογισμικού.

### II.4.I. Έλεγχος μονάδας.

Κατά τον έλεγχο μονάδας, κάθε μονάδα του λογισμικού δοκιμάζεται μεμονωμένα με σκοπό να διαπιστωθεί αν πληροί τις προδιαγραφές της. Η διαδικασία αυτή περιγράφεται σχηματικά στο Σχήμα II.7. Για τη διεκπεραίωση αυτού του επιπέδου ελέγχου μπορεί να χρησιμοποιηθεί οποιαδήποτε από τις στρατηγικές ελέγχου αναφέρθηκαν ή και οι δύο, προκειμένου να ελεγχθούν επιπλέον περιπτώσεις, με σκοπό της εξασφάλιση της ορθής λειτουργίας της.

**Σχήμα II.7** Ροή εργασιών κατά τον έλεγχο μονάδας λογισμικού.



Για την εκτέλεση του ελέγχου μονάδας χρησιμοποιούνται διάφορες τεχνικές. Οι τεχνικές αυτές διαφοροποιούνται ανάλογα με την επιλογή των περιπτώσεων ελέγχου, τα εργαλεία που μπορούν να χρησιμοποιηθούν, τη σειρά με την οποία κωδικοποιούνται και ελέγχονται οι μονάδες και με τις επιπτώσεις που έχουν στο κόστος παραγωγής των περιπτώσεων ελέγχου, καθώς και στο κόστος εντοπισμού και διόρθωσης των σφαλμάτων.

Η επιλογή της τεχνικής που θα χρησιμοποιηθεί έχει μεγάλη σημασία. Οι δύο κατηγορίες τέτοιων τεχνικών είναι η αυξητική (incremental) και η μη αυξητική (non-incremental). Ακολουθεί μια περιγραφή της μη αυξητικής τεχνικής, ενώ η περιγραφή της αυξητικής τεχνικής δίνεται μετά τον ορισμό του ελέγχου συνένωσης, καθώς στην τεχνική αυτή ολοκληρώνονται και τα δύο πρώτα επίπεδα του ελέγχου.

Στη μη αυξητική τεχνική, κάθε μονάδα ελέγχεται μεμονωμένα και εντελώς ανεξάρτητα από τις άλλες. Για τη διεκπεραίωση του ελέγχου μονάδας με αυτήν την τεχνική είναι απαραίτητη η δημιουργία καινούριων βοηθητικών μονάδων. Για κάθε μονάδα που ελέγχεται θα πρέπει να κατασκευάζεται μια βοηθητική μονάδα η οποία θα την καλεί, θα της παρέχει τα δοκιμαστικά δεδομένα εισόδου και θα τυπώνει τα αποτελέσματα της εκτέλεσης σε κάποια έξοδο. Αυτή η βοηθητική μονάδα ονομάζεται «οδηγός» (driver). Στην περίπτωση που η ελεγχόμενη μονάδα περιέχει κλήσεις σε άλλες μονάδες, τότε απαιτούνται και άλλες βοηθητικές μονάδες, οι οποίες θα πρέπει να κατασκευάζονται εκ των προτέρων, έτσι ώστε να κωδικοποιείται η επικεφαλίδα τους, να περνάνε τα ορίσματα και να επιστρέφουν κάποιες τυπικές τιμές. Αυτές οι καινούριες μονάδες ονομάζονται «στελέχη» (stubs).

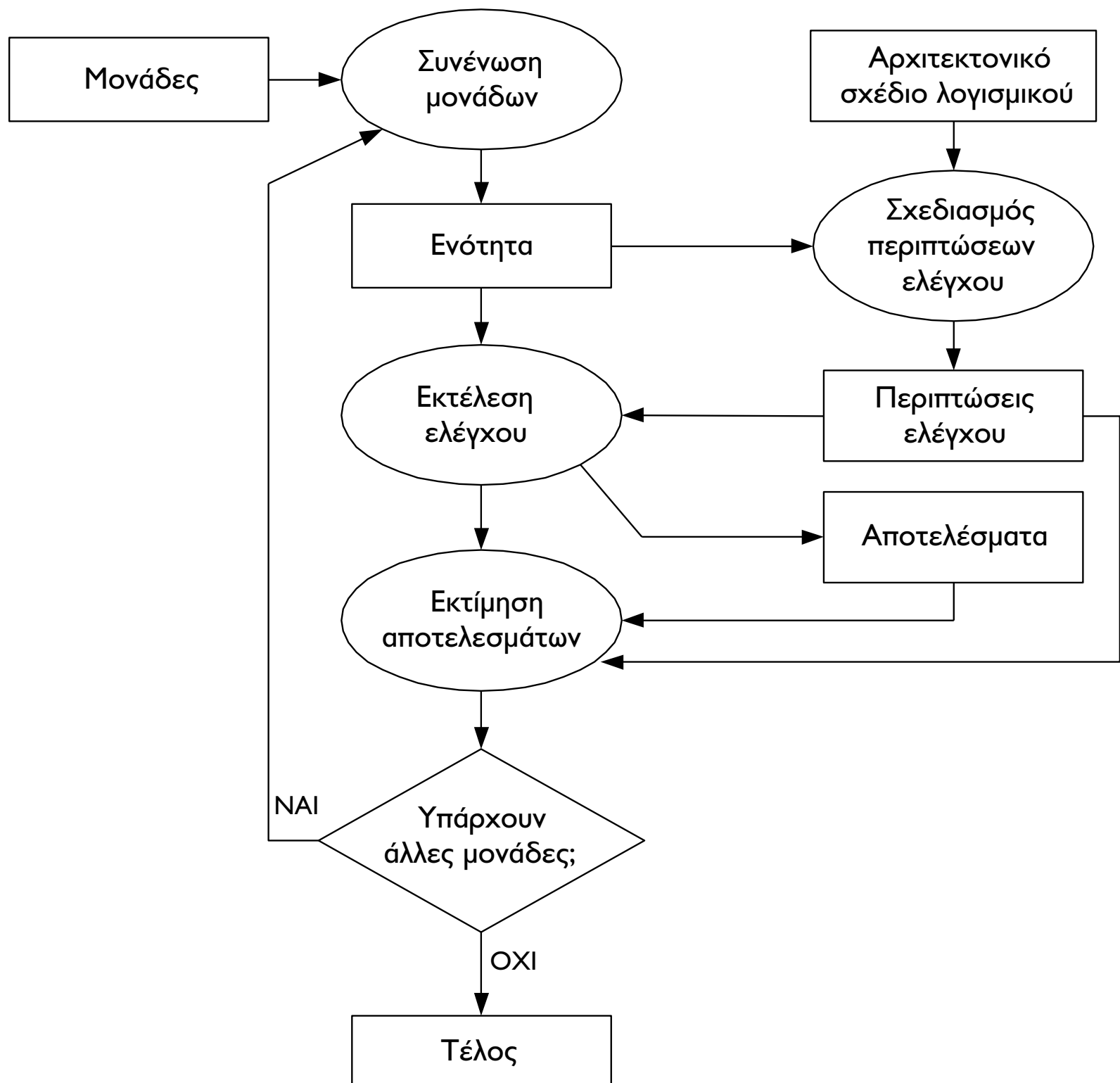
Ο έλεγχος μονάδας είναι πολύ σημαντικός και δεν θα πρέπει να παραλείπεται σε καμία περίπτωση. Μπορεί να αποκαλύψει σφάλματα τα οποία δεν είναι καθόλου εύκολο να εντοπιστούν στα επόμενα επίπεδα ελέγχου. Τέτοια μπορεί να είναι η χρησιμοποίηση κακών προγραμματιστικών τεχνικών (όπως η χρήση της εντολής goto), οι κακές επιδόσεις κ.ά.

## II.4.2. Έλεγχος συνένωσης.

Ο έλεγχος συνένωσης (integration testing) είναι η διαδικασία κατά την οποία ελέγχονται σύνθετα τμήματα, μέρη του υπό ανάπτυξη λογισμικού. Ένα τέτοιο τμήμα περιλαμβάνει περισσότερες από μία μονάδες. Αυτό που ενδιαφέρει σε αυτό το στάδιο του ελέγχου είναι να διαπιστωθεί αν οι μονάδες λογισμικού συνεργάζονται ομαλά κατά την ικανοποίηση κάποιας λειτουργικής απαίτησης.

Για τη σχεδίαση των περιπτώσεων ελέγχου στον έλεγχο συνένωσης χρησιμοποιείται και η στρατηγική του άσπρου κουτιού αλλά και η στρατηγική του μαύρου κουτιού. Ο τρόπος με τον οποίο έχουν υλοποιηθεί οι διεπαφές των μονάδων που ελέγχονται επηρεάζει τη διαδικασία αυτή. Η επιλογή των απαιτούμενων περιπτώσεων ελέγχου είναι απαραίτητο να εξασφαλίζει την εκτέλεση όλων των μονάδων του ελεγχόμενου πακέτου, καθώς και όλων των εντολών κλήσεων που περιέχονται στις μονάδες αυτές, τουλάχιστον μία φορά. Απαραίτητα λοιπόν έγγραφα για τη διεκπεραίωση του ελέγχου συνένωσης είναι το αρχιτεκτονικό σχέδιο, καθώς και ο κώδικας και το λεπτομερές σχέδιο των μονάδων. Μια σχηματική περιγραφή αυτού του επιπέδου του ελέγχου παρατίθεται στο Σχήμα II.8.

**Σχήμα II.8** Ροή εργασιών κατά τον έλεγχο συνένωσης.



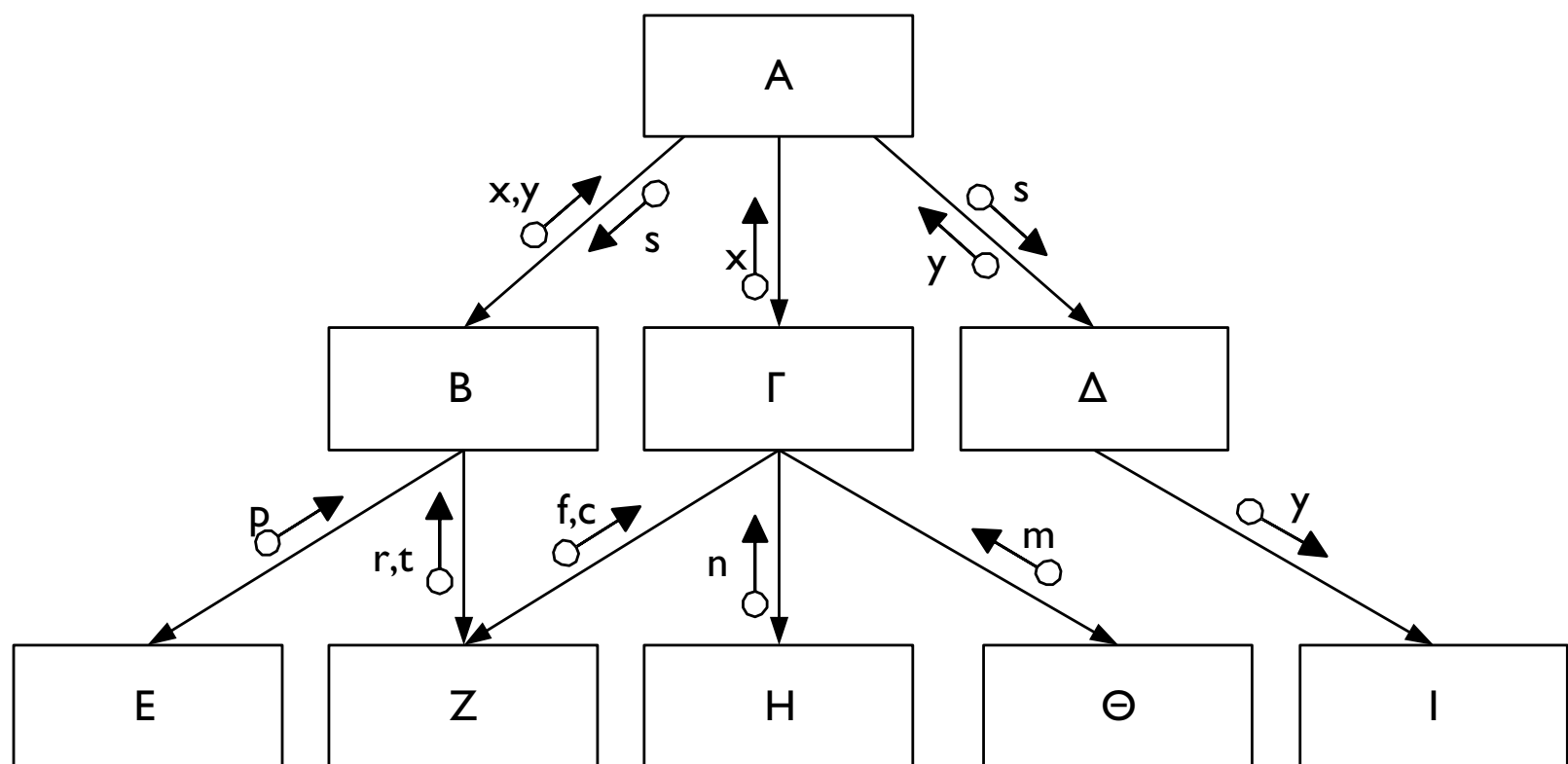
Σε περίπτωση που ο έλεγχος μονάδας έχει γίνει σωστά με τη μη αυξητική τεχνική, όπως αυτή δόθηκε στην προηγούμενη παράγραφο, τότε δεν χρειάζεται να γίνει έλεγχος συνένωσης καθώς ουσιαστικά θα έχει εκτελεστεί με τον έλεγχο μονάδας. Σε αντίθετη περίπτωση, μπορούμε να διακρίνουμε πολλές τεχνικές για την εκτέλεση του ελέγχου συνένωσης, η πιο αποτελεσματική από τις οποίες είναι η αυξητική τεχνική. Σύμφωνα με την αυξητική τεχνική, αφού κωδικοποιηθεί μια μονάδα, εκτελείται το πρώτο επίπεδο ελέγχου για αυτή (έλεγχος μονάδας). Στη συνέχεια, ενώνεται με το πακέτο στο οποίο ανήκει και εκτελείται ο έλεγχος συνένωσης. Η διαδικασία επαναλαμβάνεται για κάθε καινούρια μονάδα που κωδικοποιείται μέχρι να κατασκευαστεί ολόκληρο το λογισμικό.

Η αυξητική τεχνική μπορεί να εφαρμοστεί με δύο τρόπους: από πάνω προς τα κάτω (top-down incremental) ή από κάτω προς τα πάνω (bottom-up incremental), ανάλογα με τη σειρά με την οποία επιλέγεται να ελεγχθούν οι μονάδες του λογισμικού. Με αναφορά στο διάγραμμα δομής προγράμματος, στην περίπτωση που η κατασκευή του πηγαίου κώδικα των μονάδων αρχίσει από την κορυφή και συνεχίσει προς τα κάτω, έχουμε την προσέγγιση από πάνω προς τα κάτω, ενώ, στην αντίθετη περίπτωση, έχουμε την προσέγγιση από κάτω προς τα πάνω.

## Παράδειγμα 2/Κεφάλαιο II

Ας θεωρήσουμε το διάγραμμα δομής προγράμματος που φαίνεται στο Σχήμα II.9.

**Σχήμα II.9** Ένα διάγραμμα δομής προγράμματος.





Μια σειρά εκτέλεσης ελέγχου με την αυξητική τεχνική από πάνω προς τα κάτω είναι η ακόλουθη:

Βήμα	Υποσύστημα που ελέγχεται
1	A
2	AB
3	ABΓ
4	ABΓΔ
5	ABΓΔΕ
6	ABΓΔΕΖ
7	ABΓΔΕΖΗ
8	ABΓΔΕΖΗΘ
9	ABΓΔΕΖΗΘΙ

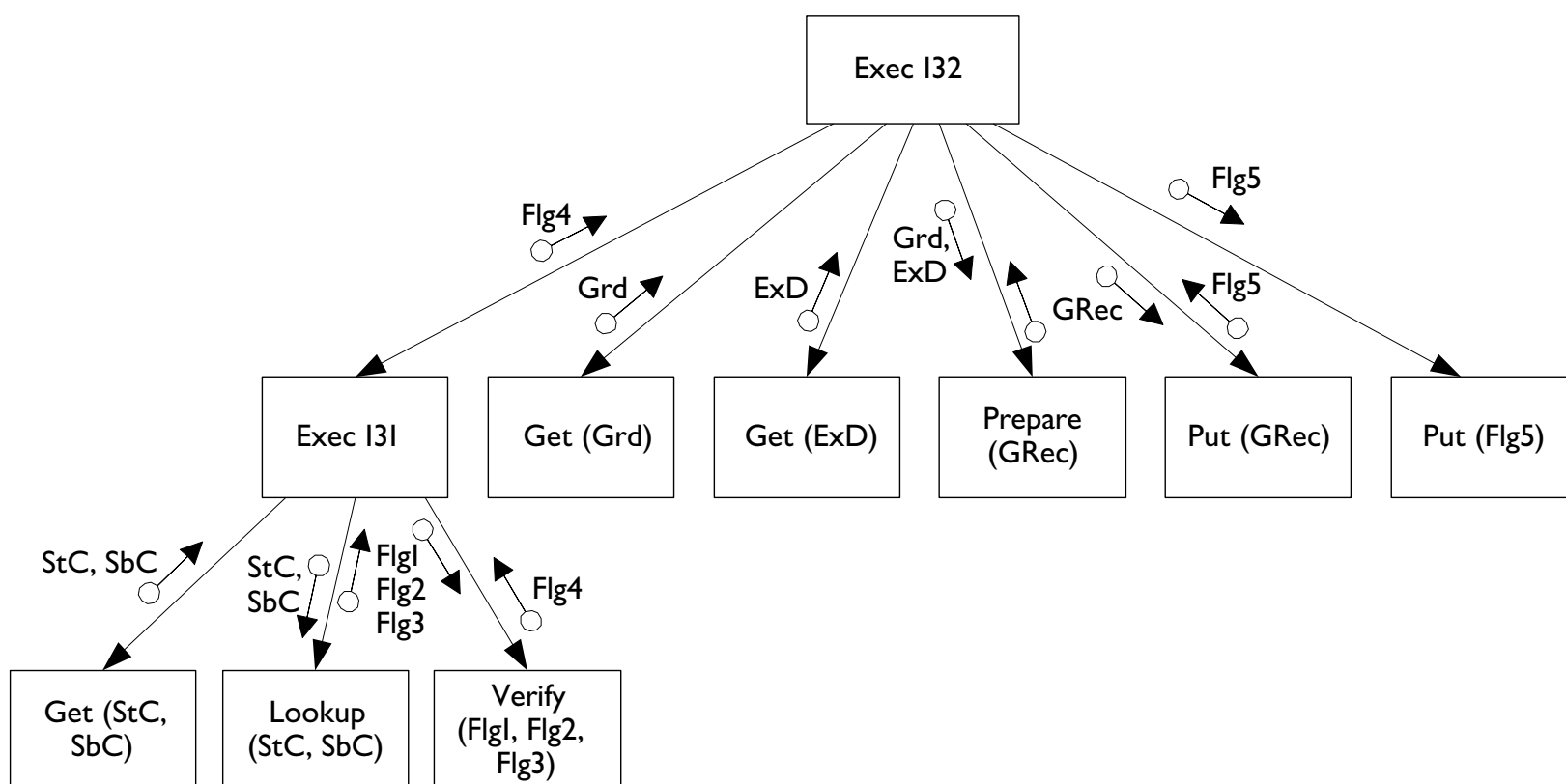
**Δραστηριότητα 4/Κεφάλαιο II**

Για τις μονάδες που περιέχονται στο διάγραμμα δομής προγράμματος που φαίνεται στο Σχήμα II.9, δώστε μια σειρά εκτέλεσης ελέγχου με την αυξητική τεχνική από κάτω προς τα πάνω.

**Δραστηριότητα 5/Κεφάλαιο II**

Ας επανέλθουμε στην εφαρμογή λογισμικού «Επίκουρος» μελετώντας το διάγραμμα δομής προγράμματος που φαίνεται στο Σχήμα II.10.

**Σχήμα II.10** Ένα διάγραμμα δομής προγράμματος από την εφαρμογή λογισμικού «Επίκουρος».



Για το διάγραμμα αυτό, δώστε τη σειρά εκτέλεσης ελέγχου σύμφωνα με την αυξητική τεχνική α) από πάνω προς τα κάτω και β) από κάτω προς τα πάνω.

## Πλεονεκτήματα και μειονεκτήματα

Ο τρόπος εκτέλεσης των δύο πρώτων επιπέδων του ελέγχου με την αυξητική από πάνω προς τα κάτω τεχνική έχει το μειονέκτημα ότι πρέπει να κατασκευάζονται στελέχη για τις μονάδες του λογισμικού που ακόμη δεν έχουν κατασκευαστεί. Από την άλλη όμως, έχει πολλά πλεονεκτήματα, όπως:

- Ο έλεγχος συνένωσης και ο έλεγχος μονάδας εκτελούνται με πολύ πιο βολικό και αξιόπιστο τρόπο. Για παράδειγμα, όταν μια μονάδα κωδικοποιηθεί και ελεγχθεί κατά τα δυο πρώτα επίπεδα ελέγχου, τότε υπάρχουν πολύ λίγα πιθανά σημεία εμφάνισης σφαλμάτων στις διεπαφές μεταξύ των μονάδων. Εφόσον το πακέτο πριν από την ένωση της τελευταίας μονάδας έχει ελεγχθεί διεξοδικά, κάποιο σφάλμα που θα αποκαλυφθεί θα πρέπει να βρίσκεται σε κάποια από τις διεπαφές της τελευταίας μονάδας.
- Είναι εφικτή η διαθεσιμότητα εκδόσεων του υπό ανάπτυξη λογισμικού κατά τη φάση του ελέγχου, στις οποίες είναι δυνατή η εκτέλεση μόνο ενός μέρους του συνόλου των λειτουργιών που προδιαγράφονται για το σύστημα. Αυτό είναι ένα σημαντικό γεγονός και όντως πολλές εταιρείες παραδίδουν το λογισμικό τους με έναν αριθμό διαδοχικών τέτοιων εκδόσεων. Η λογική που ακολουθείται είναι στην πρώτη από αυτές τις εκδόσεις του συστήματος που παραδίδεται να έχουν υλοποιηθεί οι σημαντικότερες από τις λειτουργίες που πρέπει να εκτελεί το λογισμικό και σε καθεμία από τις επόμενες εκδόσεις να προστίθενται όλο και περισσότερες.

Το βασικότερο πλεονέκτημα της προσέγγισης από κάτω προς τα πάνω είναι ότι τα σφάλματα στις μονάδες που βρίσκονται στα χαμηλότερα επίπεδα του γράφου κλήσης είναι εύκολο να αποκαλυφθούν. Από την άλλη, το βασικότερο μειονέκτημα είναι ότι για την ορθή εκτέλεσή της είναι απαραίτητες μονάδες λογισμικού που εξομοιώνουν τις συνθήκες εκτέλεσης των ελεγχόμενων μονάδων, οι οποίες είναι σαφώς πιο πολύπλοκες από τους οδηγούς προγράμματος στους οποίους αναφερθήκαμε παραπάνω. Αφού

ολοκληρωθεί και ο έλεγχος συνένωσης, περνάμε στο επόμενο επίπεδο του ελέγχου το οποίο είναι ο έλεγχος συστήματος.

## Άσκηση 2/Κεφάλαιο II

Περιγράψτε σε 5-10 γραμμές το σημαντικότερο πλεονέκτημα της αυξητικής από κάτω προς τα πάνω μεθόδου ελέγχου συνένωσης.

### II.4.3. Έλεγχος συστήματος.

Στον έλεγχο συστήματος η οντότητα που ελέγχεται είναι, πλέον, ολόκληρη η εφαρμογή λογισμικού. Στο επίπεδο αυτό του ελέγχου εκτελούνται δοκιμές και γίνονται διάφορες επιδείξεις και αναλύσεις με σκοπό να διαπιστωθεί αν το σύστημα που κατασκευάστηκε πληροί τις προδιαγραφές του. Δεν πρέπει να ξεχνάμε ότι ο στόχος σε αυτήν τη φάση ανάπτυξης του συστήματος είναι πάντα η αποκάλυψη σφαλμάτων.

Σε αυτό το επίπεδο ελέγχου χρησιμοποιείται η στρατηγική του μαύρου κουτιού για την επιλογή των περιπτώσεων ελέγχου. Οι περιπτώσεις αυτές θα πρέπει να είναι τέτοιες, ώστε να ελέγχονται όλες οι απαιτήσεις που περιγράφονται στο έγγραφο προδιαγραφών των απαιτήσεων από το λογισμικό, δηλαδή όλες οι λειτουργικές και οι μη λειτουργικές απαιτήσεις. Για την ικανοποίηση ή όχι των λειτουργικών απαιτήσεων, οι έλεγχοι καθορίζονται από τον χαρακτήρα των εκάστοτε απαιτήσεων. Για τις μη λειτουργικές απαιτήσεις, χαρακτηριστικοί έλεγχοι που εκτελούνται είναι οι παρακάτω:

- **Έλεγχοι επίδοσης (Performance testing).** Εδώ το σύστημα δοκιμάζεται με ένα σταθερό φορτίο, με σκοπό τον συντονισμό και τη βελτιστοποίηση των επιδόσεων του λογισμικού. Οι μετρήσεις που γίνονται συνήθως περιλαμβάνουν τον αριθμό των συναλλαγών, τον αριθμό των χρηστών, το μέγεθος των βάσεων δεδομένων στις οποίες γίνεται πρόσβαση κ.ά.

- **Έλεγχοι πίεσης (Stress testing).** Στις δοκιμές αυτές ελέγχονται διάφορες λειτουργίες του λογισμικού κατά την εμφάνιση ανώμαλων συνθηκών στην εκτέλεση του συστήματος. Μια τέτοια συνθήκη ορίζει την στιγμιαία εμφάνιση ενός υψηλού φορτίου όπως, για παράδειγμα, την ταυτόχρονη εκδήλωση πολλών γεγονότων σε ένα σύστημα αυτομάτου ελέγχου, καθώς και την έλλειψη μνήμης ή διαφόρων υλικών πόρων του συστήματος.
- **Έλεγχοι όγκου δεδομένων (Volume testing).** Σε αυτές τις δοκιμές ελέγχεται η δυνατότητα του λογισμικού να δουλεύει με υψηλά φορτία, τα οποία όμως δεν εμφανίζονται στιγμιαία. Αυτοί οι μεγάλοι όγκοι δεδομένων μπορεί να είναι είτε δεδομένα εισόδου, είτε δεδομένα εξόδου, είτε δεδομένα τα οποία είναι ήδη εγκατεστημένα σε κάποια βάση δεδομένων του λογισμικού. Ένα παράδειγμα ενός τέτοιου ελέγχου σε μια εφαρμογή λογισμικού τραπεζικών συναλλαγών είναι η πραγματοποίηση συναλλαγών από ολόκληρο τον πληθυσμό μιας χώρας.
- **Έλεγχοι ασφαλείας (Security testing).** Οι δοκιμές αυτές ελέγχουν κατά πόσο οι πληροφορίες που περιέχει ένα λογισμικό σύστημα είναι προσβάσιμες από άτομα στα οποία δεν θα πρέπει να παρέχεται δυνατότητα πρόσβασης στο σύστημα.
- **Έλεγχοι ανάκτησης (Recovery testing).** Στις δοκιμές αυτές ελέγχεται η δυνατότητα του λογισμικού να ανακτή τη λειτουργία του ύστερα από αιφνίδια σταματήματα. Για την εκτέλεση τέτοιων ελέγχων, το λογισμικό σύστημα εξαναγκάζεται σε αποτυχία με διάφορους τρόπους και επιβεβαιώνεται κατά πόσο είναι ικανό να ανακτή την πλήρη λειτουργία του.

#### II.4.4. Έλεγχος αποδοχής.

Ο έλεγχος αποδοχής είναι ένα υποσύνολο του ελέγχου συστήματος. Ακολουθείται δηλαδή η ίδια λογική, και τα απαραίτητα έγγραφα για τη διεκπεραίωσή του είναι επίσης οι προδιαγραφές των απαιτήσεων από το λογισμικό. Στην περίπτωση αυτή, οι περιπτώσεις ελέγχου είναι ένα υποσύνολο των αντιστοίχων περιπτώσεων του ελέγχου συστήματος, και για την επιλογή

τους είναι αποκλειστικά υπεύθυνος ο πελάτης, στις εγκαταστάσεις του οποίου μπορεί και να πραγματοποιείται ο έλεγχος αποδοχής. Ο στόχος αυτού του επιπέδου ελέγχου είναι να πειστεί ο πελάτης ότι το λογισμικό που κατασκευάστηκε για λογαριασμό του πληροί τις προδιαγραφές που τέθηκαν, η δε διάρκεια αυτού του ελέγχου μπορεί να είναι αρκετά μεγάλη, μέχρις ότου ικανοποιηθεί ο πελάτης. Στην πράξη, δεν είναι λίγες οι περιπτώσεις όπου ο έλεγχος συστήματος ταυτίζεται με τον έλεγχο αποδοχής. Τότε, ισχύει μία από τις τρεις ακόλουθες περιπτώσεις: α) ο έλεγχος συστήματος δεν είναι πλήρης, β) ο έλεγχος αποδοχής είναι ιδιαίτερα επίπονος για τον πελάτη ή γ) αυτό που εκτελείται βρίσκεται κάπου στο ενδιάμεσο και τα αποτελέσματά του δεν είναι επαρκή και αξιόπιστα.

### Άσκηση 3/Κεφάλαιο II

**Περιγράψτε σε 5-10 γραμμές αν ο έλεγχος συστήματος μπορεί να συγχωνευτεί με τον έλεγχο αποδοχής. Αν ναι, με ποιες προϋποθέσεις; Αν όχι, γιατί;**

## ΕΝΟΤΗΤΑ 11.5. Ο ΕΛΕΓΧΟΣ ΣΤΗΝ ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗ ΑΝΑΠΤΥΞΗ ΛΟΓΙΣΜΙΚΟΥ

Ανάμεσα στον δομημένο και στον αντικειμενοστρεφή προγραμματισμό υπάρχουν σημαντικές διαφορές που αφορούν την οργάνωση και έκβαση του ελέγχου. Αυτές οι διαφορές είναι οι εξής:

- i. Στη δομημένη προσέγγιση υπάρχει σαφής διαφοροποίηση μεταξύ προγραμματιστικών μονάδων (συναρτήσεις και διαδικασίες) και συνόλων που αποτελούνται από τέτοιες μονάδες (βιβλιοθήκες συναρτήσεων κ.λπ.). Στην αντικειμενοστρεφή προσέγγιση δεν υπάρχουν τόσο διακριτές διαφοροποιήσεις, καθώς η πολυπλοκότητα ενός αντικειμένου μπορεί να διαφοροποιείται σημαντικά από τη μία περίπτωση στην άλλη. Συνεπώς, ενώ στην πρώτη περίπτωση μπορεί να είναι απλούστερο να καταλήξουμε στην ορθότητα μιας μονάδας, στη δεύτερη ίσως είναι δυσκολότερο.
- ii. Το πρώτο σημείο που αναφέραμε οδηγεί σε μεγάλες εφαρμογές στην έλλειψη ξεκάθαρης ιεραρχίας και αλληλεξάρτησης αντικειμένων (στην αντικειμενοστρεφή προσέγγιση). Αυτό είναι κάτι που δυσχεραίνει τον έλεγχο κατά την ολοκλήρωση, αφού η πολυπλοκότητα του συστήματος είναι αυξημένη.

Τα δύο αυτά σημεία οδηγούν στο συμπέρασμα πως κατά την αντικειμενοστρεφή προσέγγιση ο διαχωρισμός του ελέγχου ενός συστατικού λογισμικού (module) και της ολοκλήρωσης υποσυστημάτων δεν είναι εύκολος. Συνεπώς, μπορούμε να πούμε πως ο έλεγχος αποτελεί, χωρίς διαφοροποίηση, τη συνέχεια της αντικειμενοστρεφούς ανάλυσης, σχεδίασης και υλοποίησης, με την έννοια ότι και αυτός έχει επαναληπτικό και επαυξητικό χαρακτήρα.

Είναι χαρακτηριστικό ενός ιδεατού κόσμου πως τα λάθη στην υλοποίηση συναρτήσεων, μεθόδων ή προγραμματιστικών δομών ανακαλύπτονται και επιδιορθώνονται κατά τα αρχικά στάδια του ελέγχου. Αντίστοιχα, σφάλματα και αστοχίες, για παράδειγμα στις διεπαφές, ανακαλύπτονται και επιδιορθώνονται κατά τον έλεγχο της ολοκλήρωσης της εφαρμογής κ.ο.κ. Στην



πραγματικότητα όμως, είναι σύνηθες τα σφάλματα των μονάδων να έρχονται στην επιφάνεια κατά τα προχωρημένα στάδια του ελέγχου ή και αρκετά αργότερα. Αυτό σηματοδοτεί την ανάγκη για πολλαπλούς ελέγχους όλων των εμπλεκόμενων μερών. Έτσι, προηγούμενοι έλεγχοι που έχουν ήδη ολοκληρωθεί ίσως χρειαστεί να ξαναγίνουν, καινούρια σφάλματα να προκύψουν κ.λπ. Άρα, το προηγούμενο συμπέρασμά μας, πως η διαδικασία του ελέγχου έχει έντονα επαυξητικό χαρακτήρα, επαληθεύεται.

Έχοντας ως δεδομένα πως τα περισσότερα σφάλματα εμφανίζονται κατά τα τελευταία βήματα της υλοποίησης και της παράδοσης του συστήματος και πως ο χρόνος για δραστικές αλλαγές σε αυτά τα στάδια είναι πολύ περιορισμένος, είναι εύκολο να κατανοήσουμε πως μια ενιαία και προσεκτικά σχεδιασμένη στρατηγική ελέγχου της εφαρμογής είναι ύψιστης σημασίας για την επιτυχία του έργου.

### II.5.1. Αστοχία (Defect)

Αστοχία (defect ή bug) είναι οποιοδήποτε λάθος στο λογισμικό που το εμποδίζει να εκπληρώσει τον σκοπό του, δηλαδή τις λειτουργικές ανάγκες, όπως αυτές έχουν καταγραφεί κατά την ανάλυση. Οι αστοχίες αυτές, ανάλογα με τη φύση τους, μπορούν να διαχωριστούν στις εξής κατηγορίες:

- Αστοχίες λογικής (semantic errors). Ενώ ο πηγαίος κώδικας είναι συντακτικά σωστός, το πρόγραμμα παρουσιάζει λανθασμένη συμπεριφορά, δηλαδή ενεργεί διαφορετικά απ' ό,τι θα περιμέναμε ή έχουμε προδιαγράψει. Αυτές οι αστοχίες συνήθως οφείλονται σε παρεξήγηση των προδιαγραφών από προγραμματιστές ή σε ελλιπή καταγραφή των λειτουργικών αναγκών του συστήματος.
- Αστοχίες υλοποίησης. Παρουσιάζεται λανθασμένη συμπεριφορά του κώδικα είτε κατά τη μεταγλώττιση (compile-time error) είτε κατά την εκτέλεση (run-time error). Οι αστοχίες κατά τη μεταγλώττιση είναι συνήθως εύκολο να εντοπιστούν και να διορθωθούν, τουλάχιστον έτσι ώστε ο κώδικας να μπορεί να μεταγλωττιστεί. Αντίθετα, η διόρθωση των αστοχιών που παρουσιάζονται κατά την εκτέλεση της εφαρμογής ή του υποσυστήματος μπορεί να ποικίλει σε δυσκολία. Οι αστοχίες υλοποίησης διαχωρίζονται περεταίρω:



- ο σε αστοχίες που οφείλονται στην ύπαρξη προβληματικού κώδικα και
- ο σε αστοχίες που οφείλονται στην απουσία κώδικα που θα διόρθωνε το λάθος.

Σύγχρονοι μεταγλωττιστές (compilers) και εργαλεία ελέγχου (debuggers) παρέχουν στον προγραμματιστή χρήσιμες πληροφορίες σχετικά με τον τύπο της αστοχίας, όπως επίσης και το τμήμα του κώδικα που την προκάλεσε. Τότε, ο προγραμματιστής προβαίνει στις απαραίτητες ενέργειες, προκειμένου να αποκατασταθεί η λειτουργική κατάσταση του προγράμματος. Όμως, οι ενέργειες αυτές (πρόσθεση, αφαίρεση ή αλλαγή του πηγαίου κώδικα) μπορεί να επηρεάσουν άλλα τμήματα του κώδικα με τρόπο όχι ιδιαίτερα φανερό. Αυτό μπορεί να έχει ως αποτέλεσμα τη δημιουργία περισσότερων αστοχιών σε άλλα μέρη του κώδικα, οι οποίες θα φανερωθούν σε μεταγενέστερες εκτελέσεις του προγράμματος. Προκειμένου να προλάβουμε τέτοιου είδους καταστάσεις, όπου διορθώνοντας ένα λάθος προκύπτουν περισσότερα, είμαστε αναγκασμένοι να οργανώσουμε και να εκτελέσουμε αναδρομικό έλεγχο (regression – regressive testing). Αυτός ο έλεγχος μπορεί να εφαρμοστεί σε διάφορα επίπεδα του ελέγχου κατά τη διάρκεια αλλά και αφού έχει ολοκληρωθεί η διαδικασία της υλοποίησης. Για παράδειγμα, ένας επιτυχημένος έλεγχος στο επίπεδο της ολοκλήρωσης ενός υποσυστήματος μπορεί να σηματοδοτήσει μια σειρά αναδρομικών ελέγχων μέχρι και το επίπεδο ελέγχου συναρτήσεων στο ίδιο υποσύστημα. Τέτοιοι αναδρομικοί έλεγχοι συνήθως τεκμηριώνονται στις περιπτώσεις ελέγχου, στο πλάνο ελέγχου και σε άλλα σχετικά διοικητικά/οργανωτικά κείμενα, όπως επίσης και μέσα από βοηθητικά εργαλεία που μπορεί να χρησιμοποιούνται από τους προγραμματιστές κατά την υλοποίηση.

## II.5.2. Περίπτωση ελέγχου (Test Case)

Με τον όρο «περίπτωση ελέγχου» αναφερόμαστε σε ένα σύνολο παραδοχών και μεταβλητών μέσα από το οποίο μπορεί να οριστεί –και άρα να ελεγχθεί– μια λειτουργική ανάγκη ή μια περίπτωση χρήσης της εφαρμογής. Όπως ήδη γνωρίζουμε από προηγούμενα κεφάλαια, μια λειτουργική

απαίτηση περιγράφεται από μία ή περισσότερες περιπτώσεις χρήσης, οι οποίες προδιαγράφονται αναλυτικά στο Κεφάλαιο 2. Η καθεμία από αυτές τις περιπτώσεις χρήσης απαιτεί μία ή περισσότερες περιπτώσεις ελέγχου προκειμένου να ελεγχθεί. Αναλόγως με τις επιδόσεις των περιπτώσεων ελέγχου μιας λειτουργικής απαίτησης, ο ελεγκτής λογισμικού μπορεί να καταλήξει στο συμπέρασμα πως η εν λόγω απαίτηση έχει ικανοποιηθεί ή όχι, όπως επίσης και σε ποιο βαθμό αυτό έχει επιτευχθεί. Για παράδειγμα, αν έχουν ικανοποιηθεί δύο από τις τρεις περιπτώσεις χρήσης –μέσω τις επιτυχημένης διεκπεραίωσης των περιπτώσεων ελέγχου τους– μιας λειτουργικής απαίτησης, αυτή η απαίτηση έχει ικανοποιηθεί μερικώς, έως ότου διεκπεραιωθούν επιτυχώς και οι περιπτώσεις ελέγχου της υπολειπόμενης περίπτωσης χρήσης. Από τα παραπάνω μπορούμε να καταλάβουμε πως η κάθε περίπτωση χρήσης μπορεί να ελεγχθεί από τουλάχιστον μία περίπτωση ελέγχου και, αντίστροφα, πως η κάθε περίπτωση ελέγχου μπορεί να αφορά περισσότερες από μία περιπτώσεις χρήσης. Αυτή η πληροφορία διατηρείται από τον ελεγκτή στη μορφή ενός πίνακα παρακολούθησης (traceability matrix). Ένας τέτοιος πίνακας φαίνεται στο Σχήμα II.11.

Μία περίπτωση ελέγχου προδιαγράφεται βάσει τριών κύριων χαρακτηριστικών της:

- Προαπαιτήσεις: οι προαπαιτήσεις είναι μία σειρά διαδικασιών ή αναγκαίων προϋποθέσεων που θα πρέπει να υπάρχουν, έτσι ώστε να μπορέσει να διεκπεραιωθεί η περίπτωση ελέγχου. Στις προαπαιτήσεις μπορεί επίσης να τεκμηριωθούν και οι συνθήκες κάτω από τις οποίες λαμβάνει χώρα ο έλεγχος (π.χ. κατάσταση λογισμικού ή υλικού, κατάσταση προαπαιτούμενων περιπτώσεων ελέγχου κ.λπ.).
- Τιμές εισόδου: ένα σύνολο τιμών εισόδου που αποτελεί αντιπροσωπευτικό δείγμα (με την ευρεία έννοια του όρου) του συνόλου των τιμών που αναμένεται να χρησιμοποιηθούν. Είναι κατανοητό πως αν δεν είμαστε σε θέση να ορίσουμε τις τιμές εισόδου ή να ορίσουμε επαρκώς τις προαπαιτήσεις, δεν μπορεί να προχωρήσει ο έλεγχος.
- Τιμές εξόδου: οι αναμενόμενες ή επιθυμητές τιμές εξόδου ή η συμπεριφορά για τις δεδομένες τιμές εισόδου, όπως επίσης και οι

πραγματικές τιμές εξόδου, εφόσον έχει εκτελεστεί ο έλεγχος. Για να ελεγχθεί η ορθότητα μιας περίπτωσης χρήσης μπορεί να χρειαστούν περισσότεροι από ένας διακριτοί έλεγχοι. Για παράδειγμα, ο έλεγχος ενός υποσυστήματος σύνδεσης χρηστών πρέπει να ελεγχθεί και με υπαρκτούς και με μη υπαρκτούς χρήστες.

Ανάγκη		Απ. 1 [1.1]	Απ. 1 [1.2]	Απ. 1 [1.3]	Απ. 2 [2.1]	Απ. 2 [2.2]	Απ. 2 [2.3]	Απ. 2 [2.4]	Απ. 2 [2.5]
Περίπτωση Ελέγχου	Σύνολο								
1.1.1	2	•		•					
1.1.2	1		•						
1.2.1	2		•	•					
2.1.1	1				•				
2.1.2	2				•	•			
2.1.3	2				•		•		
2.2.1	2							•	•

**Σχήμα 11.11** Παράδειγμα πίνακα παρακολούθησης των περιπτώσεων ελέγχου ενός έργου.

Στην πρώτη στήλη έχουμε τις προδιαγεγραμμένες περιπτώσεις ελέγχου. Η δεύτερη στήλη είναι προαιρετική και περιέχει το άθροισμα των περιπτώσεων χρήσης που εμπλέκονται με τη συγκεκριμένη περίπτωση ελέγχου. Οι υπόλοιπες στήλες περιέχουν τις λειτουργικές απαιτήσεις της εφαρμογής (Απ. 1, Απ. 2 κ.λπ.), όπως επίσης και τις αντίστοιχες περιπτώσεις χρήσης ([1.1], [1.2] κ.λπ.). Έτσι, μπορούμε κάθε στιγμή να επιβεβαιώσουμε ποιες περιπτώσεις ελέγχου πρέπει να διεκπεραιωθούν, προκειμένου να πιστοποιήσουμε κάποια περίπτωση χρήσης, όπως επίσης και ποιες περιπτώσεις χρήσης ελέγχονται από κάποια περίπτωση ελέγχου.

Οι περιπτώσεις ελέγχου συνήθως προδιαγράφονται ως εξής:

- **Γενικές πληροφορίες**

- ο Μοναδικός κωδικός της περίπτωσης ελέγχου.
- ο Δημιουργός (ελεγκτής, project manager κ.λπ.).
- ο Έκδοση.
- ο Σύντομος τίτλος.
- ο Σκοπός: μικρή περιγραφή του σκοπού για τον οποίο έχει γραφτεί η περίπτωση ελέγχου, ποιες περιπτώσεις χρήσης ή λειτουργικές απαιτήσεις ελέγχει.
- ο Προαπαιτούμενα: ποιες λειτουργίες ή προϋποθέσεις πρέπει να υφίστανται, έτσι ώστε να μπορεί να διεκπεραιωθεί η συγκεκριμένη περίπτωση ελέγχου.

- **Δραστηριότητες**

- ο Περιβάλλον ελέγχου: πληροφορίες σχετικά με το πώς πρέπει να έχει διαμορφωθεί το λογισμικό και το υλικό μέρος της εφαρμογής, έτσι ώστε να διεκπεραιωθεί ο έλεγχος.
- ο Αρχικοποίηση: ενέργειες που πρέπει να γίνουν πριν ξεκινήσει ο έλεγχος (για παράδειγμα, το άνοιγμα κάποιου αρχείου).
- ο Τελικό κλείσιμο (finalization): διαδικασίες που πρέπει να γίνουν μετά το τέλος του ελέγχου. Για παράδειγμα, σε περίπτωση αστοχίας κάποιου τμήματος της εφαρμογής, ο ελεγκτής θα πρέπει πρώτα να το αποκαταστήσει προτού προχωρήσει σε επόμενο έλεγχο.
- ο Ενέργειες: βήματα που πρέπει να δίνουν για να διεκπεραιωθεί ο έλεγχος.
- ο Δεδομένα εισόδου: περιγραφή των δεδομένων εισόδου που αφορούν τη συγκεκριμένη περίπτωση ελέγχου. Μερικές φορές είναι σκόπιμο να συμπεριλαμβάνουμε και κάποια αιτίαση για τη χρήση των συγκεκριμένων δεδομένων.

- **Αποτελέσματα**

- ο Αναμενόμενα αποτελέσματα: περιγραφή των αποτελεσμάτων ή των δεδομένων εξόδου που αναμένεται να δει ο ελεγκτής σύμφωνα με τη λειτουργία της περίπτωσης χρήσης ή λειτουργίας που ελέγχεται.

- ο Πραγματικά αποτελέσματα: περιγραφή των αποτελεσμάτων που ο χρήστης πραγματικά είδε μετά την ολοκλήρωση του ελέγχου. Πολλές φορές αυτό μεταφράζεται σε επιτυχία ή αποτυχία (Pass/Fail) του ελέγχου. Στην περίπτωση της αποτυχίας, καλό είναι να αναγράφεται και το ακριβές σημείο του λογισμικού που την προκάλεσε.

Οι περιπτώσεις ελέγχου, εξαιτίας της αναλυτικής τους μορφής και της άμεσης συνάφειάς τους με τις περιπτώσεις χρήσης, είναι συνηθισμένο να χρησιμοποιούνται κατά τον έλεγχο αποδοχής μιας εφαρμογής από τους τελικούς της χρήστες ή αποδέκτες. Με αυτόν τον τρόπο επαληθεύεται η τήρηση της συμφωνίας (ή σύμβασης, συμβολαίου κ.λπ.) μεταξύ του πελάτη και του αναδόχου της εφαρμογής και ακολουθεί η παραλαβή ή μη του τελικού προϊόντος.

## ΠΑΡΑΔΕΙΓΜΑ

Ας επιστρέψουμε στην εφαρμογή «Επίκουρος» και στην προδιαγεγραμμένη περίπτωση χρήσης «διαγραφή σπουδαστή». Χρησιμοποιώντας το παραπάνω πρότυπο, θα προδιαγράψουμε την αντίστοιχη περίπτωση ελέγχου.

A/A	Τίτλος	Έκδοση	Δημιουργός
01	Διαγραφή σπουδαστή μέσω της εφαρμογής «Επίκουρος»	0.1 (2/1/2007)	Νίκος Παπαδόπουλος

## Σκοπός

Ο έλεγχος της ορθής λειτουργίας της εφαρμογής «Επίκουρος» κατά τη διαγραφή ενός σπουδαστή από τη γραμματεία. Η περίπτωση ελέγχου συνδέεται με την περίπτωση χρήσης I0 της αναφοράς λειτουργικών απαιτήσεων.

## Προαπαιτούμενα

- Να υπάρχει τουλάχιστον ένας σπουδαστής στο αρχείο σπουδαστών της εφαρμογής ο οποίος να μην είναι εγγεγραμμένος σε κάποιο μάθημα.

## **Περιβάλλον ελέγχου**

- Η εφαρμογή της γραμματείας να βρίσκεται σε λειτουργία.
- Ο διακομιστής του αρχείου των σπουδαστών να βρίσκεται σε λειτουργία.
- Η μεταξύ τους σύνδεση να έχει επιτευχθεί.

## **Αρχικοποίηση**

- Το αρχείο σπουδαστών να είναι προσπελάσιμο (συμβαίνει αυτόματα κατά την έναρξη του διακομιστή του αρχείου).

## **Τελικό κλείσιμο**

- —

## **Ενέργειες**

- Ο χειριστής της γραμματείας να επιλέξει την επιλογή «διαγραφή» από το κεντρικό μενού της εφαρμογής. Στη συνέχεια, να επικυρώσει τη διαγραφή ενός από τους σπουδαστές της λίστας που θα ακολουθήσει.

## **Δεδομένα εισόδου**

- Ένας από τους σπουδαστές της λίστας διαγραφής που θα δημιουργήσει η εφαρμογή.

## **Αποτελέσματα**

Αναμενόμενα αποτελέσματα	Πραγματικά αποτελέσματα	
<p>Ο επιλεγμένος σπουδαστής πρέπει να διαγραφεί από το αρχείο και να μην είναι πλέον διαθέσιμος σε οποιαδήποτε έκφανση της εφαρμογής «Επίκουρος».</p> <p>Αυτό μπορεί να πιστοποιηθεί είτε μέσω της λίστας των σπουδαστών της εφαρμογής είτε, σε περιβάλλον ανάπτυξης, μέσω της αδυναμίας απευθείας ανάκτησης του εν λόγω σπουδαστή από το αρχείο.</p>	Επιτυχία (Pass)	<input type="radio"/>
	Αποτυχία (Fail)	<input type="radio"/>
	Άλλο (επεξήγηση):	<input type="radio"/>



### II.5.3. Πλάνο ελέγχου (Test Plan)

Το πλάνο ελέγχου είναι ένα διοικητικό έγγραφο που περιγράφει με συστηματικό τρόπο την προσέγγιση που θα ακολουθηθεί κατά τον έλεγχο ενός συστήματος. Συνήθως, ένα πλάνο ελέγχου περιέχει μια λεπτομερή περιγραφή των βημάτων του ελέγχου που θα πρέπει να γίνουν σε διάφορα στάδια της ανάπτυξης. Επίσης, ορίζει χρονικούς και ποιοτικούς στόχους που θα πρέπει να επιτευχθούν, μεταξύ άλλων. Υπάρχουν διάφοροι τρόποι σύνταξης ενός πλάνου ελέγχου όμως, επειδή πρέπει να είναι λεπτομερές αλλά ταυτόχρονα κατανοητό από τις διάφορες εμπλεκόμενες ομάδες μηχανικών (ελεγκτές, προγραμματιστές, επικεφαλής, πελάτες, ανάλογα με το σκοπό του και τις ανάγκες του έργου), συνήθως χρησιμοποιούνται πρότυπα, όπως αυτό του IEEE 829. Η ακριβής περιγραφή αυτού του προτύπου ξεφεύγει από τους στόχους του βιβλίου και έτσι δεν θα προχωρήσουμε σε αυτή –ο αναγνώστης ενθαρρύνεται να αναζητήσει περαιτέρω πληροφορίες στο διαδίκτυο.

Το πλάνο ελέγχου μπορεί να καταλήξει να γίνει πολύπλοκο και δύσκολο στη συντήρηση και ανανέωσή του και μπορεί να πάρει διάφορες μορφές ανάλογα με την τελική του χρήση. Για παράδειγμα, αν ένα πλάνο ελέγχου συντάσσεται με μοναδικό σκοπό την εσωτερική χρήση και την καθοδήγηση και παρακολούθηση της διαδικασίας του ελέγχου μιας μεγάλης εφαρμογής, μπορεί να είναι πιο τεχνικό και ουσιαστικό αλλά και πιο δυσνόητο. Σε αυτήν την περίπτωση, μπορεί επίσης να κριθεί σκόπιμο να μην ανανεώνεται συχνά ή να μην ακολουθείται κατά γράμμα, παρά μόνο στις μεγάλες αλλαγές ή κατά την ολοκλήρωση των υποσυστημάτων. Σε αυτήν την περίπτωση, το πλάνο ελέγχου αποτελεί ένα εργαλείο για την ανάπτυξη του προϊόντος. Αυτό μπορεί να κάνει το πλάνο πιο λειτουργικό για τις εσωτερικές ομάδες εργασίας αλλά, ταυτόχρονα, το καθιστά ακατάλληλο για εξωτερική χρήση, για τους τελικούς χρήστες, για παράδειγμα. Στην περίπτωση που το πλάνο ελέγχου συντάσσεται ως προϊόν κυρίως για επικοινωνιακή χρήση με εξωτερικούς παράγοντες, απαιτείται μια πιο επεξηγηματική, λιγότερο τεχνική αλλά εξίσου λεπτομερή προσέγγιση στη σύνταξή του.



Άρα, είναι κατανοητό πως η σύνταξη και η συντήρηση ενός πλάνου ελέγχου είναι χρονοβόρα και επίπονη και η ακριβής μέθοδος που θα ακολουθηθεί εξαρτάται από την προτιθέμενη τελική του χρήση. Με άλλα λόγια, η εφαρμογή του έγκειται στην ερώτηση: «Χρειαζόμαστε ένα πλάνο ελέγχου ως εργαλείο ή ως προϊόν;». Η σύγκυση μεταξύ των δύο κοστίζει σε χρόνο, σε ευελιξία και, τελικά, σε χρήμα. Στην περίπτωση που δεν είμαστε σίγουροι για την ακριβή του χρήση, ίσως είναι σκόπιμο να την αποφύγουμε τελείως.

#### **II.5.4. Αξιολόγηση ελέγχου (Test Evaluation)**

Η αξιολόγηση ελέγχου, όπως προκύπτει από τα παραπάνω, γίνεται σε διάφορα στάδια της ανάπτυξης και σε διάφορα επίπεδα του λογισμικού. Από τον έλεγχο των μονάδων, δηλαδή των μικρότερων δυνατών ελεγχόμενων τμημάτων της εφαρμογής, μέχρι τον τελικό έλεγχο αποδοχής μιας εφαρμογής, μπορεί να προκύψουν αστοχίες που να οδηγήσουν σε επανέλεγχο κώδικα χαμηλότερου επιπέδου. Συνήθως, αναλόγως με τις περιπτώσεις ελέγχου που έχουν αναγνωριστεί βάσει των περιπτώσεων χρήσης μιας εφαρμογής, τα διάφορα μέρη του προγράμματος ελέγχονται σε διάφορα χρονικά σημεία. Σε αυτά τα σημεία ελέγχεται πως τα προκαθορισμένα για την τρέχουσα φάση λειτουργικά χαρακτηριστικά της εφαρμογής παράγουν την επιθυμητή συμπεριφορά. Αν ο έλεγχος καταφέρει να αναδείξει αστοχίες, τότε το σφάλμα πρέπει να διερευνηθεί και να διορθωθεί και το παρόν τμήμα του κώδικα (όπως και όλα εκείνα που μπορεί να επηρεαστούν από την αλλαγή) να επανελεγχθεί. Αντίθετα, αν ο έλεγχος δεν αποκαλύψει αστοχίες, τότε ο κώδικας υπό έλεγχο λέμε ότι συμπεριφέρεται με τον επιθυμητό τρόπο για τη συγκεκριμένη περίπτωση ελέγχου και προχωρούμε στον επόμενο κύκλο υλοποίησης και ελέγχου του κώδικα.

Ο σκοπός των περαιτέρω κύκλων υλοποίησης και ελέγχου είναι να καλύψουν επιπρόσθετα επιθυμητά χαρακτηριστικά της εφαρμογής, έως ότου να καλυφθούν όλες οι περιπτώσεις χρήσης. Στο τέλος κάθε κύκλου, ανάλογα με τις ανάγκες του έργου, μπορεί να σηματοδοτηθεί και η τρέχουσα έκδοση της μονάδας και της εφαρμογής, εάν πρόκειται για ελέγχους ολοκλήρωσης. Για παράδειγμα, η έκδοση 0.2 ενός τμήματος κώδικα ή μιας βιβλιοθήκης θα καλύπτει ένα μικρότερο μέρος των επιθυμητών λειτουργιών της

από την έκδοση 0.5 κ.ο.κ. Είναι καλή πρακτική τέτοιες ενδιάμεσες εκδόσεις να αντικατοπτρίζουν τον μερικό έλεγχο του κώδικα έναντι απαιτήσεων και περιπτώσεων χρήσης και να συνοδεύονται από την αντίστοιχη τεκμηρίωση τόσο για τους προγραμματιστές όσο και για τους χρήστες (για παράδειγμα, στην περίπτωση έργων ανοιχτού λογισμικού).

Όταν έχουν ολοκληρωθεί όλοι οι απαραίτητοι έλεγχοι επιτυχημένα, έτσι ώστε να μπορεί να λεχθεί με σιγουριά ότι ικανοποιούνται οι περιπτώσεις χρήσης, η εφαρμογή, συνήθως, περνάει από διάφορα τελικά στάδια προτού φτάσει στην τελική της μορφή και γίνει η τελική αποδοχή εκ μέρους των πελατών ή χρηστών. Κατά τη διάρκεια αυτών των σταδίων είναι συνήθης πρακτική η εφαρμογή να έχει ήδη εγκατασταθεί και να χρησιμοποιείται από κάποιους από τους τελικούς χρήστες οι οποίοι, με τη σειρά τους, μεταφέρουν τυχόν προβλήματα στην ομάδα ελέγχου για διόρθωση. Αφότου έχει περάσει το διάστημα αυτό, το οποίο μερικές φορές αναφέρεται και ως πιλοτική φάση, και αφού η εφαρμογή έχει περάσει επιτυχημένα και τους τελευταίους ελέγχους τελικής αποδοχής, το σύστημα είναι έτοιμο να δοθεί στους τελικούς χρήστες.

Σημείωση: Στα τελευταία στάδια ανάπτυξης μιας εφαρμογής συχνά συναντάμε εκδόσεις με κωδικοποιημένα ονόματα όπως «άλφα – alpha», «βήτα – beta», «υποψήφιο για κυκλοφορία n – release candidate n», «κυκλοφορία X – release X». Τέτοιες ονομασίες χρησιμοποιούνται ευρέως κατά παραδοχή για να σηματοδοτήσουν τις τελικές φάσεις ανάπτυξης μιας εφαρμογής.

## ΕΝΟΤΗΤΑ 11.6. ΒΗΜΑΤΑ ΕΛΕΓΧΟΥ ΣΤΗΝ ΕΝΟΠΟΙΗΜΕΝΗ ΠΡΟΣΕΓΓΙΣΗ

Στην ενότητα αυτή, καθώς και στην επόμενη, θα δούμε αναλυτικότερα τα βήματα κατά την εκτέλεση του ελέγχου λογισμικού σύμφωνα με την ενοποιημένη προσέγγιση.

### 11.6.1. Δημιουργία πλάνου ελέγχου

Το πλάνο ελέγχου είναι ένα διοικητικό έγγραφο. Εναλλακτικά, μπορούμε να το σκεφτούμε ως έναν ειδικό τύπο χρονικού προγράμματος όπου αναγράφονται οι στόχοι ελέγχου. Αυτοί οι στόχοι, όπως είναι κατανοητό, είναι άμεσα συνδεδεμένοι με τα παραδοτέα του έργου και τους χρονικούς τους ορίζοντες. Όπως έχουμε δει, ανάλογα με την προτιθέμενη χρήση του πλάνου ελέγχου, ορίζουμε και τα ακριβή περιεχόμενά του. Κατά βάση, το πλάνο ελέγχου περιέχει ένα σύνολο από παραπομπές σε περιπτώσεις ελέγχου. Η λεπτομέρεια και το επίπεδο αφαίρεσης της περιγραφής αυτών των περιπτώσεων ελέγχου μέσα στο πλάνο θα εξαρτηθούν από την προτιθέμενη χρήση του. Όπως σε κάθε πλάνο, έτσι και στο πλάνο ελέγχου, η βασική πληροφορία που παρουσιάζεται είναι οι προβλεπόμενοι χρόνοι ολοκλήρωσης των διαφόρων διαδικασιών –στην προκειμένη περίπτωση των προδιαγεγραμμένων ελέγχων– και τι είδους δράσεις θα απαιτηθούν σε περίπτωση αποτυχίας των ελέγχων, δηλαδή μία τουλάχιστον στοιχειώδης διαχείριση του ρίσκου αποτυχίας.

Τα βήματα που μπορούμε να ακολουθήσουμε για να συντάξουμε ένα πλάνο ελέγχου έχοντας κατά νου τις έννοιες της αντικειμενοστρεφούς ανάπτυξης λογισμικού είναι τα ακόλουθα:

- I. Για κάθε περίπτωση χρήσης δημιουργούμε μία σειρά περιπτώσεων ελέγχου. Υπενθυμίζουμε πως κάθε περίπτωση ελέγχου μπορεί να αφορά περισσότερες από μία περιπτώσεις χρήσης και το αντίστροφο. Επιπλέον, η κάθε περίπτωση ελέγχου μπορεί να συμπεριλαμβάνει τον έλεγχο διαφόρων τμημάτων συστατικών στοιχείων της εφαρμογής. Έτσι, για την

προδιαγραφή τους, ο ελεγκτής πρέπει να λάβει υπόψη του διάφορα τεχνουργήματα της σχεδίασης.

2. Το δεύτερο σημαντικό τμήμα ενός πλάνου ελέγχου είναι το ίδιο το χρονικό πλάνο. Για να καταλήξουμε σε ένα τέτοιο πλάνο θα πρέπει να γνωρίζουμε ή να μπορούμε να προβλέψουμε τον απαιτούμενο χρόνο υλοποίησης των διαφόρων υποσυστημάτων της εφαρμογής. Έτσι, ο χρονικός προγραμματισμός των διαφόρων ελέγχων συμβαίνει στο τέλος της υλοποίησης του κάθε εμπλεκόμενου τμήματος λογισμικού. Μιας και το πλάνο ελέγχου είναι ένα έγγραφο διαχείρισης υψηλού επιπέδου, και για τους σκοπούς αυτής της ενότητας, θα κάνουμε την παραδοχή πως για τη σύνταξή του χρειάζονται μονάχα οι χρόνοι υλοποίησης των υποσυστημάτων της εφαρμογής και όχι μικρότερων τμημάτων. Η εκτίμηση των χρόνων υλοποίησης πραγματοποιείται συνδυάζοντας την εμπειρία του σχεδιαστή/ελεγκτή με τους επιτρεπόμενους χρόνους παράδοσης της εφαρμογής, το δυναμικό προσωπικού που είναι διαθέσιμο κ.λπ.
3. Συντάσσουμε το πλάνο ελέγχου σύμφωνα με κάποιο προσυμφωνημένο πρότυπο διαχειριστικών πλάνων.

## ΜΕΛΕΤΗ ΠΕΡΙΠΤΩΣΗΣ

Θα μελετήσουμε τον τρόπο δημιουργίας ενός πλάνου ελέγχου βασισμένοι στην εφαρμογή «Επίκουρος». Πιο συγκεκριμένα, θα χρησιμοποιήσουμε την παράταξη στην οποία καταλήξαμε μετά τη μελέτη περίπτωσης της Ενότητας 9.3.3. Σε αυτήν τη μελέτη έχουμε καταλήξει στα παρακάτω συστατικά στοιχεία (components):

- Διαδικτυακός «Επίκουρος»
- Παραθυρικός «Επίκουρος»
- Ειδικές εργασίες
- Τήρηση αρχείων
- Παρακολούθηση εκπαιδευτικής εργασίας

Θα επικεντρώσουμε την προσοχή μας στην περίπτωση χρήσης «τήρηση αρχείου σπουδαστών». Ένα ολοκληρωμένο πλάνο ελέγχου θα πρέπει να

συμπεριλαμβάνει τις περισσότερες, αν όχι όλες, τις περιπτώσεις χρήσης. Τα εμπλεκόμενα συστατικά στοιχεία για την υλοποίηση αυτής της περίπτωσης χρήσης είναι:

- Ο παραθυρικός «Επίκουρος» (παραθυρική εφαρμογή)
- Οι ειδικές εργασίες (βιβλιοθήκη)
- Η τήρηση αρχείων (εκτελέσιμη διαδικασία)

Επιπλέον, αν συμβουλευτούμε την προδιαγραφή αυτής της περίπτωσης χρήσης που περιγράφεται στη μελέτη περίπτωσης της Ενότητας 8.3, μπορούμε να καταλήξουμε στις παρακάτω περιπτώσεις ελέγχου:

A/A	Στόχος	Προαπαιτήσεις	Τιμές εισόδου	Τιμές εξόδου
I	Έλεγχος φόρμας εισόδου	<ul style="list-style-type: none"><li>• Η ζεύξη μεταξύ της γραμματοδείας και του διακομιστή λειτουργεί.</li><li>• Ο Παραθυρικός «Επίκουρος» είναι υπό εκτέλεση.</li><li>• Η «τήρηση αρχείων» είναι υπό εκτέλεση.</li></ul>	<ul style="list-style-type: none"><li>• Επιλογή προβολής της φόρμας τήρησης σπουδαστών.</li></ul>	<ul style="list-style-type: none"><li>• Η φόρμα τήρησης σπουδαστών με τα απαιτούμενα πεδία εμφανίζεται.</li></ul>
2	Έλεγχος εγγραφής σπουδαστή	<ul style="list-style-type: none"><li>• Η περίπτωση ελέγχου I έχει ολοκληρωθεί επιτυχώς.</li></ul>	<ul style="list-style-type: none"><li>• Τα στοιχεία ενός σπουδαστή εισάγονται στα σχετικά πεδία.</li></ul>	<ul style="list-style-type: none"><li>• Η «τήρηση αρχείων» στον διακομιστή περιέχει τα στοιχεία που έχουν εισαχθεί από τον χρήστη.</li></ul>

Ανάλογα με το επίπεδο στο οποίο θέλουμε να προσεγγίσουμε τον έλεγχο, μπορούμε να δημιουργήσουμε και άλλες, πιο λεπτομερείς, περιπτώσεις ελέγχου.

Παρομοίως, καταγράφουμε τις περιπτώσεις ελέγχου και για τις υπόλοιπες περιπτώσεις χρήσης. Το πλάνο ελέγχου, τελικά, θα περιέχει όλες τις περιπτώσεις χρήσης μαζί με επιπρόσθετα στοιχεία χρονικού σχεδιασμού για τις περιπτώσεις που δεν έχουν ελεγχθεί και για τα αποτελέσματα των ελέγχων που έχουν γίνει. Γενικά, εφόσον κάτι τέτοιο είναι προσυμφωνημένο, μπορούμε να προσθέτουμε και να αφαιρούμε στοιχεία από κάποιο πρότυπο πλάνο ελέγχου. Παρακάτω παραθέτουμε τα βασικά στοιχεία που θα πρέπει να έχει κάποιο απλό πλάνο ελέγχου, σύμφωνα με το IEEE 829:



- **Εισαγωγή**

- ο Στοιχεία προς έλεγχο: Εδώ γράφουμε τα στοιχεία της εφαρμογής που θα ελεγχθούν βάσει του πλάνου. Επίσης, δίνουμε παραπομπές στις περιπτώσεις ελέγχου που θα ακολουθήσουν και εξηγούμε γιατί έχουμε επιλέξει αυτές τις περιπτώσεις. Εάν δεν προβλέπεται να ελεγχθούν όλες οι περιπτώσεις χρήσεις ή όλες οι λειτουργικές απαιτήσεις της εφαρμογής, εδώ γράφουμε και ποιες δεν θα ελεγχθούν.
- ο Προσέγγιση που θα ακολουθηθεί: Εδώ αναφέρουμε ποια στρατηγική θα ακολουθήσουμε για τον έλεγχο των περιπτώσεων χρήσης που έχουμε επιλέξει. Ένα τέτοιο παράδειγμα είναι οι προσεγγίσεις του μαύρου ή του γυάλινου κουτιού
- ο Αναφορές: Εδώ αναφέρουμε κείμενα και άλλα έγγραφα που μπορεί να σχετίζονται με το πλάνο ελέγχου. Τέτοια έγγραφα μπορεί να είναι η ανάλυση των περιπτώσεων χρήσης, πλάνα υλοποίησης, πρότυπα πλάνων ελέγχου ή άλλων εγγράφων κ.ά.

- **Περιπτώσεις ελέγχου ανά περίπτωση χρήσης**

Εδώ παραθέτουμε τις περιπτώσεις ελέγχου στις οποίες έχουμε καταλήξει με τη μορφή που έχουν παρουσιαστεί παραπάνω. Επιπρόσθετα, για κάθε περίπτωση ελέγχου μπορούμε να δώσουμε και τα παρακάτω στοιχεία:

- ο Το όνομα του συγγραφέα της περίπτωσης ελέγχου.
- ο Το όνομα του υπεύθυνου για την έγκριση της περίπτωσης ελέγχου.
- ο Την ημερομηνία που εκτιμάται ότι θα έχει ολοκληρωθεί ο έλεγχος ή, σε περίπτωση που έχει ήδη ολοκληρωθεί, την ημερομηνία που ολοκληρώθηκε.

- **Ιστορικό**

Πολλές φορές παραθέτουμε ένα ιστορικό των αλλαγών που έχουν γίνει στο πλάνο ελέγχου. Αυτό συνήθως δίνεται με έναν απλό πίνακα που κάθε γραμμή του περιέχει την ημερομηνία που έγινε η αλλαγή, την έκδοση του πλάνου πάνω στην οποία έγινε η αλλαγή, κάποια περιγραφή και άλλα στοιχεία που θεωρούμε ότι μπορεί να μας φανούν χρήσιμα.

## II.6.2. Σχεδίαση ελέγχου

Με τον όρο «σχεδίαση ελέγχου» αναφερόμαστε στα βήματα αυτά που πρέπει να προβεί ο μηχανικός, έτσι ώστε να διευκολυνθεί ο έλεγχος της εφαρμογής τόσο κατά την υλοποίησή της όσο και κατά την ολοκλήρωση. Δεν υπάρχουν συγκεκριμένα βήματα που μπορεί κάποιος να ακολουθήσει, προκειμένου να σχεδιάσει τον έλεγχο σε όλα τα επίπεδα της ανάπτυξης με ενοποιημένο τρόπο. Η επιτυχημένη σχεδίαση ενός συστήματος συνήθως συνεπάγεται και τον επιτυχή έλεγχό της. Σε αυτό συμβάλλουν διάφορα ποιοτικά χαρακτηριστικά του σχεδίου της εφαρμογής. Τα πλέον σημαντικά, τα οποία βοηθούν στην υλοποίηση και στον έλεγχο, είναι τα εξής:

- Τα διάφορα δομικά στοιχεία (μέθοδοι, κλάσεις, υποσυστήματα κ.λπ.) της εφαρμογής θα πρέπει να βρίσκονται σε διάζευξη (decoupling). Με άλλα λόγια, το κάθε δομικό στοιχείο θα πρέπει να έχει μία καλά ορισμένη λειτουργία η οποία να είναι απαραίτητη και ικανή για την εφαρμογή. Αυτό οδηγεί σε άρτια ορισμένη συμπεριφορά και, έτσι, σε επαρκή έλεγχο. Για παράδειγμα, σε ένα σύστημα όπου μία κλάση είναι υπεύθυνη για τη διαχείριση δεδομένων και για τη διεπαφή με τους χρήστες, θα είναι δύσκολο να ελέγξουμε μόνο την επάρκεια της διεπαφής ή μόνο την ορθότητα της διαχείρισης των δεδομένων. Κατά συνέπεια, θα είναι δύσκολο να αποφανθούμε ότι η κλάση είναι σωστή, δηλαδή ότι πληροί τις απαιτούμενες λειτουργίες ή όχι.
- Η επικοινωνία μεταξύ των διαφόρων δομικών στοιχείων της εφαρμογής θα πρέπει να διέπεται από τρόπο κατανοητό, ο οποίος να είναι αποτέλεσμα της άρτιας ανάλυσης του πεδίου προβλήματος. Αυτή η επικοινωνία, όπως έχουμε δει, εκφράζεται με την προδιαγραφή και την υλοποίηση διεπαφών οι οποίες μπορούν να καθορίσουν τους όρους επικοινωνίας δομικών στοιχείων διαφόρων επιπέδων (από τους όρους επικοινωνίας μεμονωμένων κλάσεων μέχρι και ολόκληρης της εφαρμογής με εξωτερικές εφαρμογές και συστήματα). Η ύπαρξη τέτοιων διεπαφών θα βοηθήσει στη δημιουργία ρεαλιστικών περιπτώσεων χρήσης και στη διεξαγωγή ελέγχων μαύρου κουτιού. Επίσης, η ύπαρξη καλώς ορισμένων διεπαφών θα

διευκολύνει τον έλεγχο ολοκλήρωσης των διαφόρων υποσυστημάτων της εφαρμογής.

### II.6.3. Υλοποίηση ελέγχου

Η υλοποίηση του ελέγχου αποσκοπεί στη συγγραφή κώδικα ελέγχου, ούτως ώστε να μπορέσουν να ελεγχθούν και να πιστοποιηθούν τα διάφορα δομικά στοιχεία λογισμικού με αυτόματο τρόπο. Αυτή η προσέγγιση συνήθως περιορίζεται στον έλεγχο μονάδων (unit testing), δηλαδή στον έλεγχο μεμονωμένων μεθόδων και κλάσεων. Αυτή η προσέγγιση μπορεί να χρησιμοποιηθεί είτε ως ένα εργαλείο ελέγχου αφού έχει ολοκληρωθεί η υλοποίηση είτε ως ένας τρόπος ανάπτυξης της εφαρμογής από κάτω προς τα πάνω. Στην πράξη, η υλοποίηση ελέγχων μονάδας προηγείται της υλοποίησης της μονάδας και βασίζεται στην προηγούμενη σχεδίαση του λογισμικού, σε σχετικές διεπαφές κ.λπ. Η πρώτη φορά εκτέλεσης του ελέγχου θα αποτύχει, αφού δεν υπάρχει ακόμα η μονάδα. Ύστερα από τον αρχικό έλεγχο, η προσέγγιση υλοποίησης της μονάδας θα είναι επαναληπτική, με αρχικό μοναδικό σκοπό να περάσει ο έλεγχος. Όταν αυτό επιτευχθεί, ο προγραμματιστής μπορεί να προχωρήσει στην υλοποίηση άλλων μονάδων ή να βελτιώσει την υλοποίηση των υπάρχοντων μονάδων. Στο τέλος κάθε κύκλου υλοποίησης μιας μονάδας, ο προγραμματιστής θα μπορεί να ελέγχει την ορθότητα των διορθώσεων έναντι του σχετικού ελέγχου που θα έχει ήδη στη διάθεσή του. Η υλοποίηση ελέγχων μονάδας είναι εξίσου σημαντική και για περαιτέρω ελέγχους μεγαλύτερων δομικών στοιχείων, όπως υποσυστημάτων. Η ύπαρξή τους με οργανωμένο τρόπο συμβάλλει επίσης στον εκτενή έλεγχο ολοκλήρωσης και στη διατήρηση του λογισμικού τόσο κατά τη διάρκεια όσο και αφότου έχει ολοκληρωθεί ο τελικός έλεγχος του συστήματος, αφού διευκολύνει τον αυτοματοποιημένο αναδρομικό έλεγχο.



## ΜΕΛΕΤΗ ΠΕΡΙΠΤΩΣΗΣ

Στις διάφορες γλώσσες προγραμματισμού, η υλοποίηση ελέγχων μονάδας γίνεται μέσα σε πλαίσια ελέγχου (testing frameworks). Συνήθως, τέτοια πλαίσια υλοποιούνται μέσα σε βιβλιοθήκες, που ο προγραμματιστής μπορεί να καλέσει και να δημιουργήσει περιπτώσεις και σουίτες ελέγχου για τη μέθοδο ή την κλάση που θέλει να υλοποιήσει και να ελέγξει. Αυτοί οι έλεγχοι στηρίζονται πάνω σε υποθέσεις (assertions) που κάνουμε στην έξοδο της μονάδας με βάση τις τιμές εισόδου για ένα προκαθορισμένο σύνολο τιμών εισόδου. Έτσι, κατά τον έλεγχο κάποιας μεθόδου, μπορεί κανείς να υποθέσει πως η έξοδος της μεθόδου είναι σωστή. Εκτελώντας τον έλεγχο, αυτή η υπόθεση θα δοκιμαστεί αυτόματα και, αν αποδειχθεί αληθής, τότε η μέθοδος υπό έλεγχο είναι ορθή. Αν αποδειχθεί ψευδής, τότε χρειάζεται διόρθωση προτού υποβληθεί ξανά στον ίδιο έλεγχο.

Ας δούμε τον έλεγχο μονάδας στην Java μέσα από το πλαίσιο ελέγχου JUnit. Συγκεκριμένες λεπτομέρειες σύνταξης ή χρήσης του JUnit δεν θα μας απασχολήσουν. Είναι σημαντικό όμως να δείξουμε τη διαδικασία της υλοποίησης τόσο του ελέγχου όσο και του κώδικα που θα ελεγχθεί. Σε αυτήν την περίπτωση θα χρησιμοποιήσουμε ως παράδειγμα μία μέθοδο που υπολογίζει και επιστρέφει το παραγοντικό ενός ακέραιου αριθμού ( $n!$ ). Ξεκινούμε από την υλοποίηση της μεθόδου ελέγχου προτού υλοποιήσουμε την ίδια τη μέθοδο:

```
public void testFactorial()
{
    int n = 4;
    int expectedResult = 24;
    int result = factorial(n); /*κλήση στη μέθοδο*/
    assertEquals(expectedResult, result);
}
```

Εδώ χρησιμοποιούμε μία γνωστή περίπτωση, το  $4!$ , για να ελέγξουμε τη μέθοδό μας. Η μέθοδος «assertEquals» δηλώνει την υπόθεση που κάνουμε,

πως το αναμενόμενο αποτέλεσμα (expectedResult) θα είναι ίσο με το αποτέλεσμα που υπολόγισε η μέθοδος (result). Συνεχίζουμε στην υλοποίηση του προτύπου της μεθόδου:

```
public int factorial()  
{  
    return -1;  
}
```

Ο κώδικας αυτός δεν θα κάνει τίποτα χρήσιμο, όμως θα επιτρέψει στο πρόγραμμα να μεταγλωττιστεί. Αν τρέξουμε τον παραπάνω έλεγχο τώρα, προφανώς θα αποτύχει, μιας και η υπόθεση που περιέχει θα αποδειχθεί ψευδής ( $4! = 24 \neq -1$ ). Ελέγχουμε ότι αυτό ισχύει και προχωρούμε στην υλοποίηση της μεθόδου με μοναδικό κριτήριο να περάσει τον έλεγχο:

```
public int factorial(int n)  
{  
    int result = 1;  
    for (int i = 1; i <= n; i++)  
    {  
        result *= i;  
    }  
    return result;  
}
```

Αν εκτελέσουμε τον παραπάνω κώδικα, θα διαπιστώσουμε πως πράγματι υπολογίζει σωστά το παραγοντικό ενός ακέραιου. Έτσι, η εκτέλεση του ελέγχου στη συγκεκριμένη περίπτωση θα ολοκληρωθεί επιτυχημένα και, άρα, θα μπορούμε είτε να τροποποιήσουμε περαιτέρω τον κώδικα είτε να προχωρήσουμε στην υλοποίηση και έλεγχο κάποιου άλλου τμήματος του λογισμικού. Αν, για παράδειγμα, επιλέξουμε πως μια εναλλακτική υλοποίηση του υπολογισμού αυτού είναι προτιμότερη, μπορούμε να προχωρήσουμε

σε αυτή και, στη συνέχεια, να ελέγξουμε ξανά τη μέθοδο χρησιμοποιώντας τον ίδιο έλεγχο.

Με τη χρήση παρόμοιων τεχνικών κατά τη διάρκεια της υλοποίησης του συστήματος είμαστε ελεύθεροι να υλοποιούμε και να ελέγχουμε αυτόματα και επαυξητικά. Στην παραπάνω περίπτωση παρατηρούμε πως ο έλεγχος μπορεί να μην είναι ολοκληρωμένος, δηλαδή να μην ελέγχει μια πληθώρα αντιπροσωπευτικών ή ακραίων τιμών εισόδου. Στο συγκεκριμένο παράδειγμα θα έπρεπε να ελέγξουμε και την περίπτωση που η τιμή εισόδου είναι 0. Τέτοιες επιπρόσθετες ανάγκες ελέγχου μπορούν να συμπεριληφθούν στη μέθοδο ελέγχου αν τις γνωρίζουμε από την αρχή της υλοποίησης ή να προστεθούν αργότερα, κατά τη διάρκειά της.

### Δραστηριότητα 6/Κεφάλαιο II

Επεκτείνετε τον παραπάνω έλεγχο μονάδας «testFactorial», έτσι ώστε να συμπεριλαμβάνει και έλεγχο ακραίων τιμών. Πιο συγκεκριμένα, να επαληθεύει την ορθότητα του αλγορίθμου για τις τιμές  $v=0$  και  $v=1$ .

### Άσκηση 4/Κεφάλαιο II

Στο προηγούμενο κεφάλαιο, και στα πλαίσια της εφαρμογής «Επίκουρος», μελετήσαμε την υλοποίηση της κλάσης «Students». Με βάση τον παραπάνω έλεγχο μονάδας, γράψτε έναν καινούριο έλεγχο για τη μέθοδο «difference» της κλάσης «Students». Στην επίλυσή σας συμπεριλάβετε τις ακραίες περιπτώσεις η μία ή και οι δύο εμπλεκόμενες λίστες να είναι κενές.

#### II.6.4. Εκτέλεση ελέγχου ολοκλήρωσης

Αφού έχουμε δημιουργήσει ένα πλάνο ελέγχου με όλες τις περιπτώσεις ελέγχου που θέλουμε να συμπεριλάβουμε και αφότου έχουμε υλοποιήσει όλα τα απαιτούμενα και προδιαγεγραμμένα κατά τη σχεδίαση υποσυστήματα, μπορούμε να προβούμε στον έλεγχο ολοκλήρωσης. Κατά τον έλεγχο ολοκλήρωσης παίρνουμε τις ήδη ελεγμένες μονάδες και υποσυστήματα και αρχίζουμε να τα ενώνουμε και να τα ελέγχουμε διαδοχικά, σύμφωνα με τις προκαθορισμένες περιπτώσεις ελέγχου του πλάνου ελέγχου. Στο τέλος αυτής της διαδικασίας, και αφού έχει ολοκληρωθεί επιτυχώς ο έλεγχος της ολοκλήρωσης, έχουμε το τελικό σύστημα. Αυτό το σύστημα θα υποστεί τον τελικό έλεγχο συστήματος που θα δούμε στην επόμενη ενότητα, με στόχο να αποτελέσει την τελική εφαρμογή. Υπάρχουν διάφορες προσεγγίσεις στον έλεγχο της ολοκλήρωσης, όπως η προσέγγιση του Big Bang και του ελέγχου από κάτω προς τα πάνω (Bottom-Up).

Κατά την πρώτη προσέγγιση, όλα τα συστατικά στοιχεία ενώνονται, ώστε να δημιουργηθεί μία ολοκληρωμένη εφαρμογή. Στη συνέχεια, η εφαρμογή ελέγχεται με βάση το πλάνο ελέγχου και διορθώνεται στην πορεία. Αυτή η μέθοδος, όπου στην ουσία η ολοκλήρωση προηγείται της ενοποίησης, μπορεί να κερδίσει χρόνο στους ελεγκτές και, κατά συνέπεια, στην τελική παράδοση του προϊόντος. Όμως, στην περίπτωση που ο έλεγχος δεν είναι επαρκώς τεκμηριωμένος σε όλες τις βαθμίδες, μπορεί να επιφέρει και το αντίθετο αποτέλεσμα, και η ολοκλήρωση του συστήματος να καταλήξει σε αποτυχία.

Κατά την προσέγγιση από κάτω προς τα πάνω, τα συστατικά στοιχεία της εφαρμογής, αφού έχουν περάσει επιτυχώς των ελέγχων των μονάδων που τα απαρτίζουν, ολοκληρώνονται και ελέγχονται επαυξητικά. Μόνο όταν η τρέχουσα μερική ολοκλήρωση έχει περάσει επιτυχώς τους ελέγχους που την αφορούν και που προβλέπονται από το πλάνο ελέγχου επιδιώκεται η περαιτέρω ολοκλήρωση με επιπλέον συστατικά στοιχεία. Αυτή η προσέγγιση είναι πιο χρονοβόρα από την προσέγγιση του Big Bang, αλλά απαιτεί λιγότερη προετοιμασία και είναι πιο ρεαλιστική για πολύ μεγάλα συστήματα. Είναι πάντα ευκολότερο να παρακολουθήσουμε και να επιδιορθώσουμε

τυχόν σφάλματα σε μία μικρότερη βάση κώδικα, παρά σε ολόκληρη την εφαρμογή.

### II.6.5. Εκτέλεση ελέγχου συστήματος

Ο έλεγχος συστήματος είναι το τελευταίο βήμα προτού το σύστημα θεωρηθεί πως είναι έτοιμο για να παραδοθεί στον πελάτη. Κατά τον έλεγχο συστήματος, τα ολοκληρωμένα και ελεγμένα συστατικά στοιχεία του λογισμικού, παραταγμένα στο προβλεπόμενο υλικό (hardware), ελέγχονται ως ένα σύστημα. Από τη φύση του ο έλεγχος συστήματος ακολουθεί τη λογική του μαύρου κουτιού, αφού για τη διεκπεραίωσή του δεν χρειάζεται γνώση των εσωτερικών χαρακτηριστικών και αλγορίθμων των συνιστάμενων μερών.

Ο έλεγχος του συστήματος προσβλέπει στο να ανακαλυφθούν τυχόν αστοχίες τόσο μέσα στο ολοκληρωμένο, πλέον, λογισμικό όσο και σε σχέση με την παράταξή του στο υλικό. Για τη διεκπεραίωση αυτού του ελέγχου, ο ελεγκτής συμβουλευείται τις περιπτώσεις χρήσης, έτσι ώστε να τις ελέγξει έναντι των λειτουργικών απαιτήσεων της εφαρμογής. Επιπλέον, ο ελεγκτής συμβουλευεται τα μοντέλα παράταξης και άλλα διαχειριστικά κείμενα, έτσι ώστε να τα ελέγξει έναντι των μη λειτουργικών απαιτήσεων, των απαιτήσεων δηλαδή που έχουν να κάνουν με το υλικό και με άλλα εξωτερικά χαρακτηριστικά (π.χ. λειτουργικά συστήματα κ.ά.).

Σε αυτή τη φάση ελέγχου, η πρόθεση του ελεγκτή είναι να αναγκάσει το σύστημα να αστοχήσει είτε εξαιτίας του λογισμικού, είτε εξαιτίας του υλικού, είτε εξαιτίας ενδεχόμενων λαθών στην ολοκλήρωση των δύο. Με βάση τις λειτουργικές ανάγκες του συστήματος διερευνώνται οι πιθανότερες συμπεριφορές χρηστών και καταγράφονται οι τυχόν αστοχίες. Όταν αυτές εξαντληθούν, είναι συνηθισμένο για τον ελεγκτή να διερευνήσει πιο ακραίες συμπεριφορές χρηστών ή να εξαντλήσει τις δυνατότητες του υλικού, έτσι ώστε να προκαλέσει το σύστημα να αστοχήσει. Κατά τον έλεγχο του συστήματος ο ελεγκτής ακολουθεί την πλέον επιθετική προσέγγιση. Όταν το σύστημα έχει βρεθεί να πληροί τις προϋποθέσεις ελέγχου, όπως αυτές έχουν προβλεφθεί στο πλάνο ελέγχου, και άρα να πληροί τις λειτουργικές

αλλά και τις μη λειτουργικές απαιτήσεις, είναι έτοιμο να δοθεί στον πελάτη για τον τελικό έλεγχο αποδοχής και παραλαβή.

## ΕΝΟΤΗΤΑ 11.7. ΑΝΑΦΟΡΕΣ ΕΛΕΓΧΟΥ

Οι αναφορές σφαλμάτων είναι ένα σημαντικό τμήμα της τεκμηρίωσης του ελέγχου. Η σύνταξη αυτών των αναφορών αποτελεί το τελευταίο στάδιο της διαδικασίας του ελέγχου (Σχήμα II.3). Ο σκοπός μιας τέτοιας αναφοράς είναι να καταγράψει με λεπτομέρεια ένα γεγονός που συμβαίνει κατά την εκτέλεση του ελέγχου το οποίο υποδεικνύει την ύπαρξη ενός σφάλματος, ώστε να συμβάλλει στην αποτελεσματική διόρθωσή του. Μια τέτοια αναφορά συντάσσεται για κάθε σφάλμα το οποίο αποκαλύπτεται κατά την εκτέλεση ενός ελέγχου. Μια δομή της αναφοράς σφάλματος προτείνεται στο Σχήμα II.12.

**Σχήμα II.12** Υπόδειγμα δομής αναφοράς σφάλματος.

### Αναφορά σφάλματος

1. Ταυτότητα του εγγράφου
2. Ανακεφαλαίωση του ελέγχου που εκτελέστηκε
3. Περιγραφή του ελέγχου και του σφάλματος
4. Επιπτώσεις

Δύο ακόμη σημαντικά έγγραφα για την ολοκληρωμένη τεκμηρίωση του ελέγχου είναι το ημερολόγιο ελέγχου (Test log) και η περιληπτική έκθεση του ελέγχου (Test-Summary report). Στο ημερολόγιο ελέγχου καταγράφονται πληροφορίες σχετικές με την εκτέλεση των ελέγχων και μπορεί να φανεί ιδιαίτερα χρήσιμο κατά τη σύνταξη των αναφορών σφαλμάτων, καθώς επίσης και κατά την εκτέλεση μελλοντικών ελέγχων. Το καταλληλότερο χρονικό πλαίσιο για τη σύνταξή του είναι παράλληλα με την εκτέλεση του ελέγχου. Ακολουθώντας, στο Σχήμα II.13 παρουσιάζεται μια δομή του ημερολογίου ελέγχου.



## **Σχήμα II.13** Το ημερολόγιο ελέγχου.

### **Ημερολόγιο ελέγχου**

1. Ταυτότητα του ημερολόγιου ελέγχου
2. Περιγραφή
3. Εγγραφές δραστηριοτήτων και γεγονότων
  - 3.1 Περιγραφή εκτέλεσης οντότητας
  - 3.2 Αποτελέσματα εκτέλεσης οντότητας
  - 3.3 Πληροφορίες περιβάλλοντος
  - 3.4 Γεγονότα ανώμαλης συμπεριφοράς
  - 3.5 Ταυτότητες αναφορών σφαλμάτων

Τέλος, η περιληπτική έκθεση του ελέγχου συνοψίζει τις διαδικασίες που επιτελέστηκαν κατά την εκτέλεσή του και παρουσιάζει εκτιμήσεις για την περιεκτικότητά του. Επίσης, περιλαμβάνει και μια αξιολόγηση του όλου ελέγχου. Στο Σχήμα II.14 παρουσιάζεται μια δομή της περιληπτικής έκθεσης του ελέγχου.

## **Σχήμα II.14** Δομή περιληπτικής έκθεσης ελέγχου.

### **Περιληπτική έκθεση ελέγχου**

1. Ταυτότητα της έκθεσης
2. Περίληψη
3. Αποκλίσεις
4. Εκτίμηση περιεκτικότητας
5. Περίληψη αποτελεσμάτων
6. Αξιολόγηση
7. Περίληψη δραστηριοτήτων
8. Εγκρίνοντες

## ΕΝΟΤΗΤΑ 11.8. ΔΙΟΡΘΩΣΗ ΣΦΑΛΜΑΤΩΝ

Μέχρι τώρα, στο παρόν κεφάλαιο ασχοληθήκαμε με τον έλεγχο, δηλαδή με τη διαδικασία αποκάλυψης σφαλμάτων στο λογισμικό. Η εργασία αυτή, όμως, απλά επιβεβαιώνει την ύπαρξη τέτοιων σφαλμάτων, καθώς συνήθως δεν καθορίζει την ακριβή θέση ή την υφή τους. Το επόμενο βήμα για την ορθή ανάπτυξη του λογισμικού συστήματος είναι η διόρθωση των σφαλμάτων που βρέθηκαν κατά τον έλεγχο. Για τον σκοπό αυτό χρησιμοποιούνται ειδικές τεχνικές (debugging techniques).

Για την αποτελεσματική εφαρμογή των τεχνικών αυτών είναι απαραίτητη η συλλογή όσο το δυνατόν περισσότερων στοιχείων για το κάθε σφάλμα. Τέτοια στοιχεία μπορούν να συγκεντρωθούν με τη χρησιμοποίηση διαφόρων μεθόδων, όπως:

- Με την εισαγωγή εντολών για την εκτύπωση διαγνωστικών μηνυμάτων στον κώδικα του λογισμικού. Τέτοια μηνύματα δίνουν τις τιμές των μεταβλητών σε ορισμένα κρίσιμα σημεία του προγράμματος και επιβεβαιώνουν ότι η εκτέλεσή του έχει περάσει από τα σημεία αυτά. Αυτή η μέθοδος συγγραφής πηγαίου κώδικα ονομάζεται «αμυντικός προγραμματισμός».
- Με τη μελέτη αντιγράφων της μνήμης σε δυαδική ή συμβολική μορφή. Τέτοια αντίγραφα παρέχουν πληροφορίες για την κατάσταση της εκτέλεσης του λογισμικού σε διάφορα κρίσιμα σημεία.
- Με τη χρησιμοποίηση ιχνών (traces). Τα ίχνη καταγράφουν τις εντολές όπως αυτές εκτελούνται, καθώς και τις τιμές προκαθορισμένων μεταβλητών ανά πάσα στιγμή.
- Με την τοποθέτηση σημείων διακοπής (breakpoints) του προγράμματος. Στα σημεία αυτά διακόπτεται η εκτέλεση του προγράμματος και ο έλεγχος δίνεται στο τερματικό απ' όπου ο χρήστης μπορεί να δει τις τιμές των μεταβλητών, των καταχωρητών, να αλλάξει τις τιμές αυτές κ.λπ.

Ο έλεγχος και η διόρθωση των σφαλμάτων είναι δύο πολύ σημαντικά στάδια της ανάπτυξης ενός λογισμικού συστήματος. Σχετικές μελέτες έχουν αποδείξει ότι οι δύο αυτές εργασίες μπορεί να εξοικονομήσουν μέχρι και



το 50% του προϋπολογισμού για την κατασκευή μιας εφαρμογής λογισμικού. Η επιτυχημένη εκτέλεσή τους συμβάλλει ώστε ο εκάστοτε κατασκευαστής να κερδίζει τόσο σε χρήμα όσο και σε αξιοπιστία.

## ΕΝΟΤΗΤΑ 11.9. ΕΡΓΑΛΕΙΑ ΕΛΕΓΧΟΥ

Ο έλεγχος, ο εντοπισμός και η διόρθωση των σφαλμάτων ενός λογισμικού συστήματος είναι πολύ χρονοβόρες και μονότονες εργασίες, οι οποίες στο παρελθόν εκτελούνταν με το χέρι. Σήμερα, κατασκευάζονται ολοένα και περισσότερα εργαλεία τα οποία υποστηρίζουν ή και αυτοματοποιούν την εκτέλεσή τους, παράγοντας από μόνα τους μεγάλο μέρος της τεκμηρίωσης του ελέγχου. Η ανάπτυξη ολοένα και πολυπλοκότερων και μεγαλύτερων προγραμμάτων δεν θα ήταν δυνατή χωρίς τα εργαλεία αυτά. Οι πιο χαρακτηριστικές κατηγορίες τέτοιων εργαλείων είναι οι ακόλουθες:

- **Γεννήτριες δοκιμαστικών δεδομένων** (test data generators). Τα εργαλεία αυτά είναι προγράμματα που παράγουν δοκιμαστικά δεδομένα για τον έλεγχο μιας εφαρμογής λογισμικού. Η χρησιμοποίησή τους συνηθίζεται στα στάδια του ελέγχου συστήματος και του ελέγχου αποδοχής όπου είναι απαραίτητος μεγάλος όγκος δοκιμαστικών δεδομένων.
- **Συγκριτές αρχείων** (file comparators). Ο συγκριτής αρχείων είναι ένα εργαλείο το οποίο συγκρίνει δύο αρχεία και καταγράφει τις όποιες διαφορές τους. Ένα τέτοιο εργαλείο χρησιμοποιείται συνήθως για τη σύγκριση μεγάλου όγκου αποτελεσμάτων με τα αναμενόμενα αποτελέσματα της εκτέλεσης διαφόρων ελέγχων.
- **Εργαλεία αλλοίωσης του λογισμικού** (mutation testing tools). Τα εργαλεία αυτά εισάγουν τεχνητά λάθη σε μια οντότητα λογισμικού για τον έλεγχο της συμπεριφοράς της. Η τεχνική αυτή ελέγχου έχει αποδειχθεί ιδιαίτερα αποτελεσματική κατά το στάδιο του ελέγχου μονάδας, ενώ είναι μάλλον ακατάλληλη για τον έλεγχο συστήματος.

- **Εργαλεία ελέγχου δυναμικής ανάλυσης** (dynamic analysis testing tools). Τα εργαλεία αυτά «παρακολουθούν» την εκτέλεση μιας εφαρμογής λογισμικού και ελέγχουν τη λειτουργία της με την εισαγωγή γραμμών κώδικα που δηλώνουν ότι η εκτέλεση του προγράμματος έχει φτάσει σε ένα συγκεκριμένο σημείο. Η εισαγωγή αυτή γίνεται πριν από την εκτέλεση του υπό έλεγχο λογισμικού.
- **Εργαλεία ελέγχου στατικής ανάλυσης** (static analysis testing tools). Τα εργαλεία αυτά δεν εκτελούν το σύστημα αλλά ελέγχουν τον κώδικά του. Συνήθως χρησιμοποιούνται για την εύρεση καθαρά προγραμματιστικών σφαλμάτων, όπως η μη αρχικοποίηση κάποιας μεταβλητής.
- **Βιβλιοθηκάριοι δεδομένων ελέγχου** (test data librarians). Τα εργαλεία αυτά αποθηκεύουν διάφορα αρχεία ελέγχου και αποτελεσμάτων και είναι ικανά για αυτόματη επανεκτέλεση ελέγχων.

Επίσης, για τον έλεγχο των πιο πολύπλοκων, παράλληλων λογισμικών συστημάτων πραγματικού χρόνου έχουν αναπτυχθεί σήμερα προηγμένα εργαλεία τα οποία ονομάζονται «εξομοιωτές» (simulators). Τα εργαλεία αυτά εξομοιώνουν το πραγματικό περιβάλλον λειτουργίας του υπό έλεγχο λογισμικού συστήματος. Μπορούν, επομένως, να παράγουν πραγματικά δεδομένα για τον έλεγχο ενός συστήματος και με τη βοήθεια ορισμένων από τα εργαλεία που αναφέρθηκαν παραπάνω καθιστούν εφικτή την εξομοίωση μεγάλου πραγματικού χρόνου λειτουργίας του. Με τον τρόπο αυτό, προσδίδουν σημαντική αξιοπιστία στα αποτελέσματα του ελέγχου.

# ΕΝΟΤΗΤΑ 11.10. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΚΑΙ ΑΣΚΗΣΕΩΝ ΑΥΤΟΑΞΙΟΛΟΓΗΣΗΣ

## Δραστηριότητα I/Κεφάλαιο II

Το πλάνο του ελέγχου μπορεί να ξεκινήσει να γράφεται αρκετά νωρίς και, συγκεκριμένα, μετά την ολοκλήρωση της συγγραφής των απαιτήσεων από το λογισμικό διότι:

- (α) Είναι η στιγμή που αποφασίζεται η ενσωμάτωση στο λογισμικό χαρακτηριστικών που θα πρέπει να ελεγχθούν και για τα οποία μπορούν να ετοιμαστούν τα αναγκαία για τον έλεγχο έγγραφα.
- (β) Για ορισμένες μονάδες ή χαρακτηριστικά του λογισμικού ο έλεγχος μπορεί να ξεκινήσει πριν ολοκληρωθεί η συγγραφή του πηγαίου κώδικα και όλων των υπολοίπων, ώστε να κερδίζεται χρόνος.

## Δραστηριότητα 2/Κεφάλαιο II

Στο πρόγραμμα που δίνεται, οι μεταβλητές εισόδου είναι οι ακέραιες μεταβλητές day, month και year. Λόγω της έννοιας που αποδίδεται στις μεταβλητές αυτές (π.χ. στη μεταβλητή day αποδίδεται η έννοια της ημέρας) προκύπτουν κάποιες συνθήκες εισόδου, δηλαδή προδιαγράφονται ορισμένα διαστήματα τιμών. Σύμφωνα με αυτά τα διαστήματα τιμών καταγράφουμε τις έγκυρες αλλά και τις άκυρες κλάσεις ισοδύναμων τιμών. Έτσι έχουμε:

Μεταβλητή	Συνθήκη εισόδου	Κλάσεις ισοδύναμων τιμών	
		Έγκυρες	Άκυρες
day	$1 \leq day \leq 31$	$1 \leq day \leq 31$	$day < 1$ $day > 31$
month	$1 \leq month \leq 12$	$1 \leq month \leq 12$	$month < 1$ $month > 12$

Παρατηρήστε ότι οι συνθήκες εγκυρότητας προκύπτουν εδώ από μία καθολικά γνωστή σύμβαση, αυτή του ημερολογίου. Αυτό δεν ισχύει πάντα. Σε πολλές περιπτώσεις, οι συνθήκες εγκυρότητας των δεδομένων εισόδου προκύπτουν μέσα από το ίδιο το πρόβλημα.

Επίσης, παρατηρήστε ότι, για λόγους απλότητας, στο σημείο αυτό αναφερόμαστε μόνο στο πρώτο τμήμα του προγράμματος (γραμμές 1-8). Για να καλύψουμε όλο το πρόγραμμα, αρκεί να αντικαταστήσουμε το πάνω όριο (31) με την τιμή της μεταβλητής «UpperDayValue».

ΠΡΟΣΕΓΓΙΣΗ ΙΣΟΔΥΝΑΜΗΣ ΔΙΑΜΕΡΙΣΗΣ

Για την κατασκευή περιπτώσεων ελέγχου αρκεί να επιλέξουμε για κάθε μεταβλητή μία τιμή από κάθε κλάση ισοδύναμων τιμών της. Οπότε, ένα ενδεικτικό σύνολο περιπτώσεων ελέγχου είναι το παρακάτω:

Περίπτωση ελέγχου	Δοκιμαστικά δεδομένα εισόδου			Αναμενόμενα αποτελέσματα
Id#	day	month	year	flag
1	5	3	1950	0
2	5	13	1950	1
3	5	-2	1950	1
4	36	3	1950	1
5	-7	3	1950	1
6	-7	13	1950	1
7	-7	-2	1950	1

ΠΡΟΣΕΓΓΙΣΗ ΣΥΝΟΡΙΑΚΩΝ ΤΙΜΩΝ

Σύμφωνα με την προσέγγιση συνοριακών τιμών, για την κατασκευή των περιπτώσεων ελέγχου επιλέγουμε έγκυρες περιπτώσεις για τα άκρα του κάθε διαστήματος και άκυρες για τιμές ακριβώς έξω από τα άκρα. Οπότε, ένα ενδεικτικό σύνολο περιπτώσεων ελέγχου είναι το παρακάτω:

Περίπτωση ελέγχου	Δοκιμαστικά δεδομένα εισόδου			Αναμενόμενα αποτελέσματα
Id#	day	month	year	flag
1	1	1	1900	0
2	1	12	1900	0
3	31	1	1900	0
4	31	12	1900	0
5	1	0	1900	1
6	1	13	1900	1
7	31	0	1900	1

Επίσης, παρατηρήστε ότι άκυρα δεδομένα εισόδου, όπως για παράδειγμα «31.2.2009», εντοπίζονται από την ίδια τη μονάδα προγράμματος. Συγχαρητήρια αν δώσατε τη λύση που δώσαμε και εμείς.

### Δραστηριότητα 3/Κεφάλαιο II

Για την κάλυψη της συνθήκης που ζητείται, είναι απαραίτητη μία επιπλέον περίπτωση ελέγχου, η οποία και δίνεται παρακάτω:

Δοκιμαστικά δεδομένα			Κάλυψη εντολών (μονοπάτι εκτέλεσης)	Κάλυψη συνθηκών (Εντολές απόφασης – τιμές συνθηκών)
A	B	C		
2	2	I	I-2-3-6-I2-I8-I9	3:T-F, 6: F, I2: F, I8:T

Ο εντοπισμός πλήρων περιπτώσεων ελέγχου δεν είναι πάντα απλή υπόθεση. Ακόμα και σε ένα μικρό πρόγραμμα, όπως αυτό του παραδείγματος, μπορείτε να διαπιστώσετε ότι απαιτείται αρκετός χρόνος και προσοχή για να παρακολουθήσετε μια έτοιμη λύση. Φανταστείτε τι ισχύει στην πραγματικότητα, όπου το πλήθος και η πολυπλοκότητα των μονάδων προγράμματος είναι δραματικά μεγάλο. Τώρα, μάλλον μπορείτε να εξηγήσετε για ποιο λόγο η συγγραφή προγραμμάτων χωρίς σφάλματα είναι πολύ δύσκολη.

## Δραστηριότητα 4/Κεφάλαιο II

Μια πιθανή σειρά εκτέλεσης ελέγχου με την αυξητική από κάτω προς τα πάνω τεχνική είναι η ακόλουθη:

Βήμα	Υποσύστημα που ελέγχεται
1	Ε
2	ΕΖ
3	ΕΖΗ
4	ΕΖΗΘ
5	ΕΖΗΘΙ
6	ΕΖΗΘΙΒ
7	ΕΖΗΘΙΒΓ
8	ΕΖΗΘΙΒΓΔ
9	ΕΖΗΘΙΒΓΔΑ

Επιβεβαιώστε ότι «χτίζουμε» την πυραμίδα από τη βάση της. Συγχαρητήρια σε όσους το κατάφεραν με την πρώτη. Οι υπόλοιποι, παραπέμπονται στο παράδειγμα 2 και καλούνται να ξαναδοκιμάσουν.

Δραστηριότητα 5/Κεφάλαιο II

α) Μια σειρά εκτέλεσης ελέγχου με την αυξητική από πάνω προς τα κάτω τεχνική είναι η ακόλουθη:

Βήμα	Υποσύστημα που ελέγχεται
I	(Exec I32)
2	(Exec I32) (Exec I3I)
3	(Exec I32) (Exec I3I) (Get (Grd))
4	(Exec I32) (Exec I3I) (Get (Grd)) (Get (ExD))
5	(Exec I32) (Exec I3I) (Get (Grd)) (Get (ExD)) (Prepare (GRec))
6	(Exec I32) (Exec I3I) (Get (Grd)) (Get (ExD)) (Prepare (GRec)) (Put (GRec))
7	(Exec I32) (Exec I3I) (Get (Grd)) (Get (ExD)) (Prepare (GRec)) (Put (GRec)) (Put (Flg5))
8	(Exec I32) (Exec I3I) (Get (Grd)) (Get (ExD)) (Prepare (GRec)) (Put (GRec)) (Put (Flg5)) (Get (Stc, SbC))
9	(Exec I32) (Exec I3I) (Get (Grd)) (Get (ExD)) (Prepare (GRec)) (Put (GRec)) (Put (Flg5)) (Get (Stc, SbC)) (Lookup(StC, SbC))
10	(Exec I32) (Exec I3I) (Get (Grd)) (Get (ExD)) (Prepare (GRec)) (Put (GRec)) (Put (Flg5)) (Get (Stc, SbC)) (Lookup(StC, SbC)) (Verify (FlgI, Flg2, Flg3))



β) Μια σειρά εκτέλεσης ελέγχου με την αυξητική από κάτω προς τα πάνω τεχνική είναι η ακόλουθη:

Βήμα	Υποσύστημα που ελέγχεται
1	(Get (Stc, SbC))
2	(Get (Stc, SbC)) (Lookup(StC, SbC))
3	(Get (Stc, SbC)) (Lookup(StC, SbC)) (Verify (Flgl, Flg2, Flg3))
4	(Get (Stc, SbC)) (Lookup(StC, SbC)) (Verify (Flgl, Flg2, Flg3)) (Exec I3I)
5	(Get (Stc, SbC)) (Lookup(StC, SbC)) (Verify (Flgl, Flg2, Flg3)) (Exec I3I) (Get (Grd))
6	(Get (Stc, SbC)) (Lookup(StC, SbC)) (Verify (Flgl, Flg2, Flg3)) (Exec I3I) (Get (Grd)) (Get (ExD))
7	(Get (Stc, SbC)) (Lookup(StC, SbC)) (Verify (Flgl, Flg2, Flg3)) (Exec I3I) (Get (Grd)) (Get (ExD)) (Prepare (GRec))
8	(Get (Stc, SbC)) (Lookup(StC, SbC)) (Verify (Flgl, Flg2, Flg3)) (Exec I3I) (Get (Grd)) (Get (ExD)) (Prepare (GRec)) (Put (GRec))
9	(Get (Stc, SbC)) (Lookup(StC, SbC)) (Verify (Flgl, Flg2, Flg3)) (Exec I3I) (Get (Grd)) (Get (ExD)) (Prepare (GRec)) (Put (GRec)) (Put (Flg5))
10	(Get (Stc, SbC)) (Lookup(StC, SbC)) (Verify (Flgl, Flg2, Flg3)) (Exec I3I) (Get (Grd)) (Get (ExD)) (Prepare (GRec)) (Put (GRec)) (Put (Flg5)) (Exec I32)

Ο αναγνώστης παρακαλείται να συγχωρήσει τη δυσκολία ανάγνωσης των παραπάνω πινάκων, αλλά η σειρά κλήσης των μονάδων έπρεπε να δοθεί έτσι για λόγους οικονομίας χώρου.

## Δραστηριότητα 6/Κεφάλαιο II

Ο νέος έλεγχος μονάδας θα είναι ο εξής:

```
public void testFactorial()
{
    int n = 4;
    int expectedResult = 24;
    int result = factorial(n); /*κλήση στη μέθοδο*/
    assertEquals(expectedResult, result);

    /*επιπρόσθετοι έλεγχοι: */
    n = 0;
    expectedResult = 1;
    result = factorial(n); /*κλήση στη μέθοδο*/
    assertEquals(expectedResult, result);

    n = 1;
    expectedResult = 1;
    result = factorial(n); /*κλήση στη μέθοδο*/
    assertEquals(expectedResult, result);
}
```

# Άσκηση I/Κεφάλαιο II

Ένα σύνολο περιπτώσεων ελέγχου για την εξασφάλιση της πλήρους κάλυψης του πηγαίου κώδικα της μονάδας είναι το παρακάτω:

Δοκιμαστικά δεδομένα			Κάλυψη εντολών (μονοπάτι εκτέλεσης)
Day	month	year	
32	14	1999	1-2-3-4-5-6-7-8-9-29
30	3	1999	1-2-3-4-5-7-9-10-11-12-13-15-20-22-23-24-25-26-27-28-29
30	4	0	1-2-3-4-5-7-9-10-11-13-14-15-20-22-23-24-25-26-27-28-29
30	2	1999	1-2-3-4-5-7-9-10-11-13-15-16-18-19-20-21-28-29
30	2	2000	1-2-3-4-5-7-9-10-11-13-15-16-17-18-20-21-28-29

Δεν υπάρχει τυποποιημένος τρόπος για τον καθορισμό των ζητούμενων δεδομένων. Είναι απαραίτητο να μελετάται κάθε φορά ο πηγαίος κώδικας. Συχνά, η εργασία καθορισμού τέτοιων δεδομένων φαίνεται κουραστική, ιδιαίτερα όταν γίνεται από τον ίδιο τον συγγραφέα του πηγαίου κώδικα. Στην πραγματικότητα, είναι ένας από τους λίγους τρόπους που διαθέτουμε για να μπορούμε να υποστηρίξουμε (προσέξτε: όχι να τεκμηριώσουμε!) την μη ύπαρξη σφαλμάτων σε ένα πρόγραμμα. Η σωστή απάντηση της άσκησης σημαίνει ικανοποιητική κατανόηση του θέματος και αξίζει τα συγχαρητήριά μας.

## Άσκηση 2/Κεφάλαιο II

---

Το βασικότερο πλεονέκτημα της προσέγγισης αυτής είναι ότι τα σφάλματα στις μονάδες που βρίσκονται στα χαμηλότερα επίπεδα του διαγράμματος δομής είναι εύκολο να αποκαλυφθούν. Αυτό ισχύει καθώς οι μονάδες αυτές έχουν πιο απλή δομή και όσο πιο χαμηλό είναι το επίπεδο του διαγράμματος δομής στο οποίο βρίσκονται, τόσο λιγότερες κλήσεις σε άλλες μονάδες τείνουν να περιέχουν. Επίσης, όλες οι μονάδες τις οποίες καλούν έχουν συνήθως ήδη ελεγχθεί, με αποτέλεσμα ο εντοπισμός σφαλμάτων κατά τον έλεγχό τους να γίνεται ακόμη πιο εύκολος.

## Άσκηση 3/Κεφάλαιο II

---

Ο έλεγχος συστήματος μπορεί να συγχωνευτεί με τον έλεγχο αποδοχής, καθώς ο έλεγχος αποδοχής δεν παύει να αποτελεί ένα υποσύνολο του ελέγχου συστήματος. Απαραίτητη προϋπόθεση, βέβαια, είναι το περιβάλλον ελέγχου να είναι όσο το δυνατόν πιο πιστό αντίγραφο του προοριζόμενου περιβάλλοντος λειτουργίας του λογισμικού. Επίσης, θα πρέπει να γίνει σε συνεννόηση με τον πελάτη, έτσι ώστε η κατασκευή των περιπτώσεων ελέγχου να έχει λάβει υπόψη της τις απαιτήσεις του. Συγχαρητήρια αν δώσατε τη σωστή απάντηση! Επίσης (περισσότερα) συγχαρητήρια αν, άσχετα με την απάντηση που δώσατε, καταφέρνετε να κατασκευάζετε λογισμικό χωρίς σφάλματα ή, καλύτερα, με σφάλματα που δεν εκδηλώνονται!

Η εμπειρία δείχνει ότι πολλά έργα παραδίδονται με μοναδικό έλεγχο τον έλεγχο αποδοχής κατευθείαν από τον πελάτη. Για λόγους εντυπώσεων, συχνά ο κατασκευαστής λογισμικού δεν αποκαλύπτει ότι θα είναι ο ίδιος ο πελάτης που θα ελέγξει το λογισμικό. Με τη βοήθεια της τύχης, κάτι τέτοιο μπορεί να περάσει απαρατήρητο και να μην εκδηλωθούν σφάλματα, ιδιαίτερα σφάλματα που προκαλούν ζημιά στον πελάτη και βλάπτουν το κύρος του κατασκευαστή. Συνήθως όμως, αργά ή γρήγορα, τα σφάλματα του λογισμικού εκδηλώνονται αμείλικτα...

```
public void testDifference()
{
    Students students = new Students();
    ArrayList myList = new ArrayList();
    ArrayList resultList = null;
    int resultSize;
    int expectedResultSize;

    // και οι δύο λίστες είναι κενές
    expectedResultSize = 0;
    resultList = students.difference(myList);
    resultSize = resultList.size();
    assertEquals(expectedResultSize, resultSize);
    // η εξωτερική λίστα είναι κενή
    // προσθέτουμε ένα σπουδαστή στην λίστα μας
    students.addStudent(new Student());
    expectedResultSize = 1;
    resultList = students.difference(myList);
    resultSize = resultList.size();
    assertEquals(expectedResultSize, resultSize);
    // η εσωτερική λίστα είναι κενή
    // αδειάζουμε την εσωτερική λίστα
    students = new Students();
    // προσθέτουμε ένα σπουδαστή στην εξωτερική λίστα
    myList.add(new Student());
    expectedResultSize = 1;
    resultList = students.difference(myList);
    resultSize = resultList.size();
    assertEquals(expectedResultSize, resultSize);
}
```

```
// οι δύο λίστες είναι πανομοιότιπες
// αρχικοποιούμε τις δύο λίστες
students = new Students();
myList = new ArrayList();
// δημιουργούμε έναν υποθετικό σπουδαστή
Student testStudent = new Student();
testStudent.firstName = "John";
testStudent.lastName = "Smith";
// και τον προσθέτουμε και στις δύο λίστες
students.addStudent(testStudent);
myList.add(testStudent);
// σε αυτή την περίπτωση η διαφορά πρέπει να είναι μηδέν
expectedResultSize = 0;
resultList = students.difference(myList);
resultSize = resultList.size();
assertEquals(expectedResultSize, resultSize);
```

## ΒΙΒΛΙΟΓΡΑΦΙΑ

Howden, W. E., *Functional Program Testing and Analysis*, McGraw-Hill International Editions, 1987.

*IEEE Standard for Software Test Documentation*, ANSI/IEEE, Std 829-1991

Pressman, R. S., *Software Engineering-A Practitioners Approach*, McGraw-Hill.

Sommerville, I. *Software Engineering*, London: Addison-Wesley.

## ΔΙΟΙΚΗΣΗ ΣΧΗΜΑΤΙΣΜΩΝ ΛΟΓΙΣΜΙΚΟΥ

Σκοπός του κεφαλαίου είναι να παρουσιαστούν οι βασικές έννοιες και οι κύριες εργασίες της διοίκησης σχηματισμών λογισμικού (software configuration management), η οποία εκτελείται καθ' όλο τον κύκλο ζωής, παράλληλα με τις εργασίες ανάπτυξης λογισμικού.

Μετά τη μελέτη του κεφαλαίου αυτού, ο αναγνώστης θα είναι σε θέση:

- να περιγράφει την έννοια του σχηματισμού λογισμικού και να δίνει τρία παραδείγματα στοιχείων σχηματισμών λογισμικού,
- να διακρίνει τις εκδόσεις ενός στοιχείου σχηματισμών λογισμικού σε αναθεωρήσεις και παραλλαγές και να τις χαρακτηρίζει με παραστατικά ονόματα,
- να αναγνωρίζει και να δίνει τη γενική περιγραφή για τις τέσσερις εργασίες της διοίκησης σχηματισμών λογισμικού.

### Έννοιες-κλειδιά

---

- Διοίκηση σχηματισμών λογισμικού
- Σχηματισμός λογισμικού
- Στοιχείο σχηματισμού λογισμικού
- Έκδοση
- Αναθεώρηση
- Παραλλαγή
- Βασική γραμμή
- Αποδέσμευση



## Σύνοψη

---

Η διοίκηση σχηματισμών λογισμικού εκτελείται παράλληλα με τις εργασίες ανάπτυξης λογισμικού και δεν αφορά αυτή καθαυτή την κατασκευή των συστατικών στοιχείων λογισμικού, αλλά ασχολείται με την οργάνωση και τη διαχείρισή τους. Οι μεγάλες εφαρμογές λογισμικού, οι οποίες έχουν εκδόσεις για διαφορετικές πλατφόρμες υλικού, λειτουργικά συστήματα, γλώσσες κ.λπ., δεν είναι δυνατόν να κατασκευαστούν και να συντηρηθούν χωρίς διοίκηση σχηματισμών.

Οι βασικές εργασίες της διοίκησης σχηματισμών λογισμικού είναι ο καθορισμός των σχηματισμών και των σχέσεων μεταξύ τους, ο έλεγχος μεταβολών, ο έλεγχος ποιότητας και η αναφορά κατάστασης σχηματισμών λογισμικού. Οι εργασίες αυτές δεν είναι εύκολο να γίνονται με το χέρι και συνήθως υποστηρίζονται από εργαλεία ανάπτυξης λογισμικού. Αν και εκ πρώτης όψεως προσθέτουν κόστος (και, σύμφωνα με κάποιους, γραφειοκρατία) στην ανάπτυξη λογισμικού, συμβάλλουν στην τήρηση μιας αναγκαίας τάξης και στην καλύτερη ποιότητα του παραγόμενου λογισμικού.

## Εισαγωγικές παρατηρήσεις

---

Ανεξάρτητα από την προσέγγιση που ακολουθείται, κατά την ανάπτυξη λογισμικού παράγεται ένα μεγάλο σύνολο ενδιάμεσων προϊόντων – συστατικών στοιχείων λογισμικού. Κάθε συστατικό έχει τα δικά του χαρακτηριστικά και ανήκει σε σύνολα, καθένα από τα οποία πραγματοποιεί ή σχετίζεται με κάποιες συγκεκριμένες λειτουργίες. Επίσης, κατά τον κύκλο ζωής της εφαρμογής, τα συστατικά λογισμικού εξελίσσονται και μεταβάλλονται είτε για να ενσωματωθούν σε αυτά διορθώσεις είτε για να τροποποιηθούν κάποια χαρακτηριστικά τους. Ο κύκλος αυτός έχει γίνει ιδιαίτερα έκδηλος με τις εφαρμογές λογισμικού που τρέχουν σε smart phones, οι οποίες ενημερώνονται συνεχώς.

Μια εκδοχή του λογισμικού η οποία βγαίνει «προς τα έξω» συγκροτείται από επιλεγμένα δομικά στοιχεία, με συγκεκριμένα χαρακτηριστικά και κατάσταση. Ωστόσο, το πλήθος και η πολυπλοκότητα των συσχετίσεων μεταξύ των στοιχείων αυτών μπορούν να καταστήσουν τη διαδικασία της επιλογής και τεκμηρίωσης της «σωστής» εκδοχής πολύ δύσκολη. Προβλήματα όπως η ονοματολογία, η ομαδοποίηση, η απόδοση χαρακτηριστικών ιδιωμάτων και η διαχείριση των τροποποιήσεων

κάνουν την εμφάνισή τους και μπορούν εύκολα να δημιουργήσουν πονοκέφαλο στον κατασκευαστή.

Για τον σκοπό αυτό, παράλληλα με τις εργασίες ανάπτυξης λογισμικού και, μάλιστα, ανεξάρτητα από τη φιλοσοφία και τη συγκεκριμένη μεθοδολογία ανάπτυξης που ακολουθείται, πρέπει να τρέχει η εργασία διοίκησης σχηματισμών λογισμικού (*Software Configuration Management*), μια εισαγωγική αναφορά στην οποία επιχειρούμε στο κεφάλαιο αυτό.

## ΕΝΟΤΗΤΑ 12.1. ΒΑΣΙΚΕΣ ΕΝΝΟΙΕΣ

### 12.1.1. Η έννοια του σχηματισμού λογισμικού

Από τις πρώτες φάσεις της ανάπτυξης του λογισμικού, και πολύ περισσότερο στη φάση χρήσης – συντήρησής του, το λογισμικό εξελίσσεται και μεταβάλλεται. Αιτίες για την πραγματοποίηση μεταβολών είναι οι διορθώσεις, οι επεκτάσεις αλλά και οι μεταβαλλόμενες ανάγκες των χρηστών. Είναι, πλέον, αποδεκτό ότι το λογισμικό είναι κάτι σαν ένας ζωντανός οργανισμός που εξελίσσεται. Η πραγματοποίηση των μεταβολών και η απόκλιση ή, έστω, η εξέλιξη της αρχικής εικόνας δεν είναι εύκολη υπόθεση. Αυτό ισχύει ιδιαίτερα στην κατασκευή μεγάλων εφαρμογών, δηλαδή λογισμικού μεγάλης κλίμακας, η οποία συνήθως γίνεται από πολυπληθείς ομάδες ανάπτυξης.

Εκτός από αυτή καθαυτή την πραγματοποίηση των απαιτούμενων εργασιών ανάπτυξης λογισμικού, υπόσταση αποκτά τότε και το πρόβλημα της διαχείρισης των συστατικών στοιχείων του λογισμικού. Λέγοντας «διαχείριση» αναφερόμαστε στον χαρακτηρισμό, την ταξινόμηση, την αρχειοθέτηση και τη διάθεση συστατικών στοιχείων λογισμικού στα μέλη της ομάδας ανάπτυξης. Μερικά από τα προβλήματα που παρουσιάζονται όταν η ομάδα ανάπτυξης λογισμικού είναι πολυπληθής, είναι τα παρακάτω:

- Ταυτόχρονη ενημέρωση: Όταν δύο ή περισσότεροι προγραμματιστές εργάζονται ξεχωριστά πάνω στο ίδιο πρόγραμμα, μπορεί να προκύψουν καταστροφικές ασυμβατότητες από τις αλλαγές που κάνουν.
- Διαμοιράσιμος ή κοινός κώδικας: Όταν γίνεται μια αλλαγή σε μια μονάδα που είναι κοινή σε πολλά υποσυστήματα, όλοι οι προγραμματιστές που εμπλέκονται στην ανάπτυξη αυτών των υποσυστημάτων πρέπει να ενημερώνονται για τις ενδεχόμενες επιδράσεις που μπορεί να έχει αυτή η αλλαγή στη δική τους εργασία.
- Εκδόσεις: Οι μεταβολές στο λογισμικό πρέπει να γίνονται με ελεγχόμενο τρόπο. Συνήθως αυτές καταλήγουν σε μεγάλο αριθμό εκδόσεων (versions), ακόμα και για ένα απλό συστατικό του

λογισμικού, πράγμα που κάνει τη δομή του συστήματος εξαιρετικά πολύπλοκη.

Η εργασία της διαχείρισης των συστατικών μονάδων του λογισμικού δεν γίνεται στο χαμηλότερο επίπεδο (συναρτήσεις, διαδικασίες, κλάσεις κ.ά.), αλλά είναι πιο βολικό να γίνεται σε κάποιο πιο υψηλό επίπεδο. Περνώντας από τον ορισμό και τη δημιουργία συστατικών μονάδων λογισμικού στη διαχείρισή τους, συναντάμε την έννοια του σχηματισμού λογισμικού (configuration).

### **Σχηματισμός λογισμικού:**

Όλα τα συστατικά στοιχεία λογισμικού που κατασκευάζονται στις διάφορες φάσεις της ανάπτυξης και συγκροτούν μια εφαρμογή ονομάζονται συλλογικά «σχηματισμός λογισμικού» (software configuration).

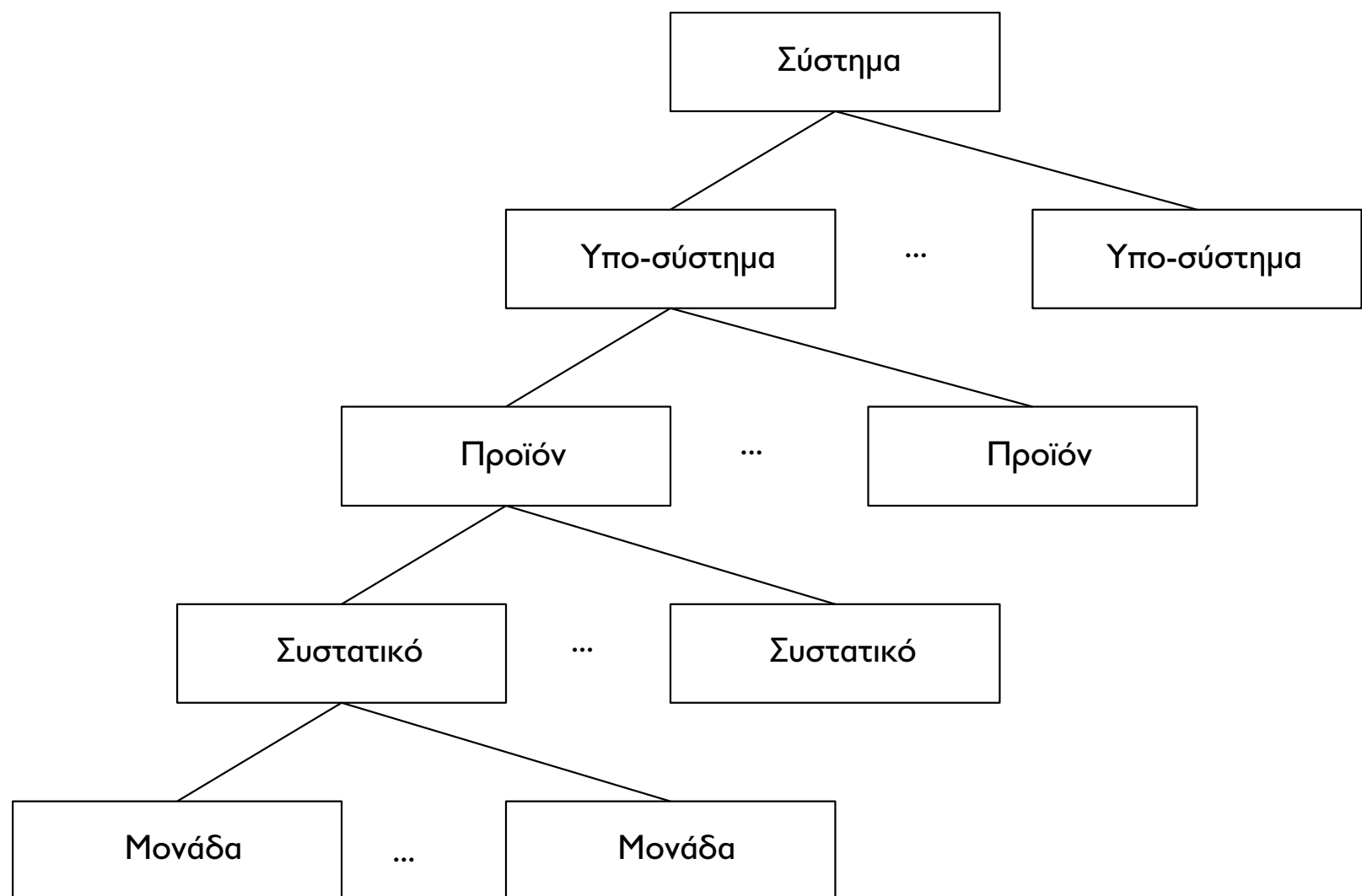
Ο όρος «συστατικά στοιχεία λογισμικού» στον προηγούμενο ορισμό μπορεί να αναφέρεται είτε σε ολόκληρα προϊόντα είτε σε τμήματα αυτών. Κάθε ξεχωριστό στοιχείο ή μια συλλογή από στοιχεία, που για διαχειριστικούς λόγους τα θεωρούμε ως μία μονάδα, καλείται «στοιχείο σχηματισμού λογισμικού».

### **Στοιχείο σχηματισμού λογισμικού:**

Ένα ατομικό συστατικό στοιχείο λογισμικού ή μια συλλογή από τέτοια στοιχεία τα οποία σχετίζονται μεταξύ τους, ώστε να συγκροτούν μία διαχειριστική οντότητα.

Έτσι, αν ένα στοιχείο είναι μια συλλογή από άλλα στοιχεία ονομάζεται «σύνθετο», ενώ διαφορετικά ονομάζεται «απλό» ή «ατομικό». Τα στοιχεία σχηματισμών λογισμικού μπορεί να συμμετέχουν σε ιεραρχίες που ορίζονται από τη σχέση «αποτελείται\_από» μεταξύ σύνθετων στοιχείων και των μερών τους (με τη σειρά τους απλών ή σύνθετων). Μια τέτοια ιεραρχία φαίνεται στο Σχήμα 12.1. Είναι φανερό ότι η διαχείριση ενός στοιχείου σχηματισμού λογισμικού είναι πιο απλή όταν ανήκει στα χαμηλότερα επίπεδα της ιεραρχίας.

**Σχήμα Ι2.Ι** Μια ιεραρχία των μερών ενός συστήματος λογισμικού βασισμένη στη σχέση «αποτελείται\_από».



## Δραστηριότητα I/Κεφάλαιο I2

Έχοντας ως αναφορά την εφαρμογή λογισμικού «Επίκουρος» που συζητήθηκε στα Κεφάλαια 4 και 5, να αναφέρετε τρία παραδείγματα στοιχείων σχηματισμού λογισμικού που, κατά τη γνώμη σας, μπορεί να θεωρηθούν απλά και τρία που μπορεί να θεωρηθούν σύνθετα.

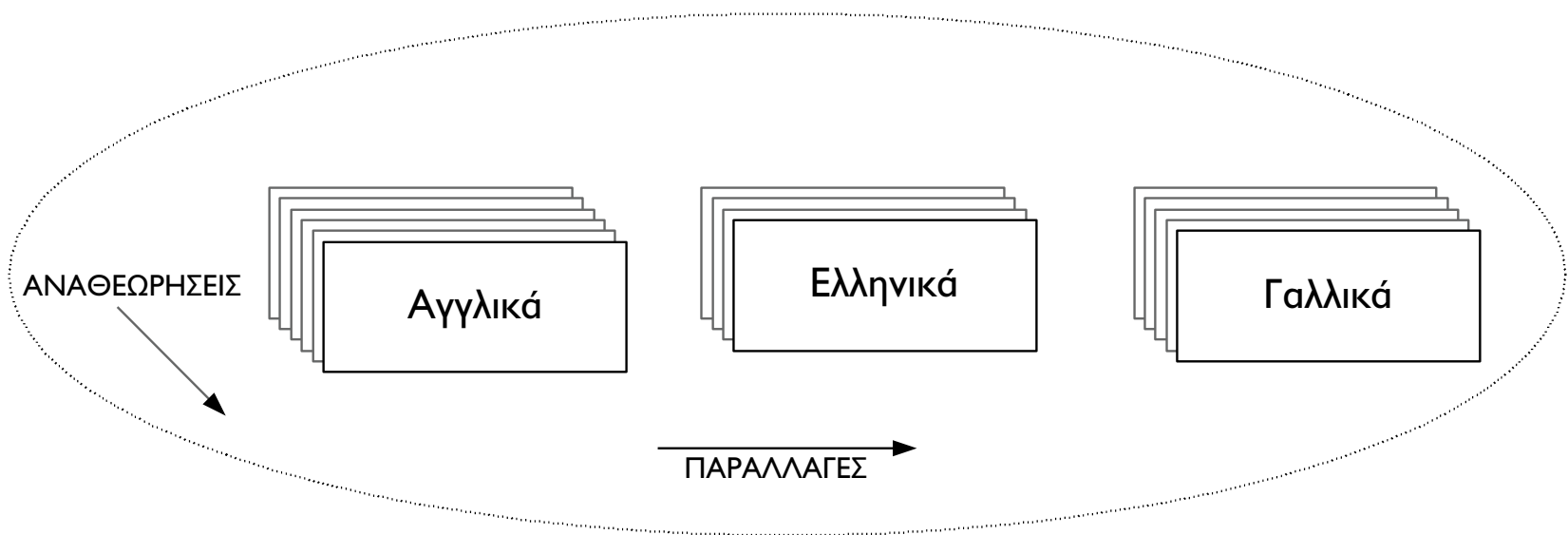
### 12.1.2. Η έννοια της βασικής γραμμής

Οι μεταβολές που πραγματοποιούνται στο λογισμικό κατά τον κύκλο ζωής του τελικά υλοποιούνται στο επίπεδο ατομικών συστατικών λογισμικού τα οποία, όπως είπαμε, συγκροτούν στοιχεία σχηματισμών λογισμικού. Κάθε μεταβολή σε κάποιο ατομικό στοιχείο δεν καταργεί απαραίτητα την προηγούμενη μορφή αυτού, αλλά ενδέχεται και να συνυπάρχει με αυτή. Για παράδειγμα, η τροποποίηση ενός συστατικού στοιχείου της διεπαφής με τον χρήστη για μια άλλη (εθνική) γλώσσα δεν καταργεί το στοιχείο που τροποποίησε, αλλά συνυπάρχει με αυτό. Μπορούμε, λοιπόν, γενικά να θεωρούμε ότι οι μεταβολές που συμβαίνουν στα συστατικά στοιχεία λογισμικού έχουν ως αποτέλεσμα αυτά να υπάρχουν σε πολλές μορφές (εκδοχές), οι οποίες ονομάζονται «εκδόσεις» (versions).

Οι εκδόσεις των απλών στοιχείων σχηματισμών λογισμικού δημιουργούν τις εκδόσεις των σύνθετων. Αν ληφθούν υπόψη οι εκδόσεις, η δομή ενός συστήματος λογισμικού γίνεται εξαιρετικά πολύπλοκη. Διακρίνουμε δύο είδη εκδόσεων: τις αναθεωρήσεις (revisions) και τις παραλλαγές (variants). Οι παραλλαγές παριστάνουν παράλληλες, ανεξάρτητες γραμμές ανάπτυξης ενός στοιχείου σχηματισμών λογισμικού καθεμία εκ των οποίων διαφέρει από την άλλη σε κάποια χαρακτηριστικά. Για παράδειγμα, παραλλαγές κάποιων στοιχείων σχηματισμών λογισμικού μιας εφαρμογής δημιουργούνται όταν αυτή τρέχει σε πολλά λειτουργικά συστήματα ή σε διαφορετικές (εθνικές) γλώσσες. Οι αναθεωρήσεις αντικαθιστούν η μία την άλλη και δημιουργούνται συνήθως κατά τη διόρθωση σφαλμάτων, κατά την πρόσθεση νέων ή κατά την τροποποίηση υπαρχουσών λειτουργιών. Ένα παράδειγμα

αναθεωρήσεων και παραλλαγών ενός στοιχείου σχηματισμών λογισμικού για διαφορετικές γλώσσες φαίνεται στο Σχήμα Ι2.2.

**Σχήμα Ι2.2** Αναθεωρήσεις και παραλλαγές.





Όταν μια έκδοση ενός στοιχείου σχηματισμών λογισμικού διανέμεται εκτός του οργανισμού ανάπτυξης, τότε αυτή καλείται «αποδέσμευση» (release). Μια αποδέσμευση δημιουργεί υποχρεώσεις του κατασκευαστή του στοιχείου απέναντι στους αποδέκτες της, οι οποίοι μπορεί να είναι τελικοί χρήστες ή άλλοι κατασκευαστές λογισμικού. Χωρίς παρακολούθηση των χαρακτηριστικών των στοιχείων σχηματισμών λογισμικού, οι συνεχείς μεταβολές μπορεί ταχύτατα να οδηγήσουν σε χάος. Για τον λόγο αυτό, χρειάζεται ένας μηχανισμός που να τις χαρακτηρίζει και να τις θέτει κάτω από έλεγχο. Ένας τέτοιος μηχανισμός είναι η βασική γραμμή (baseline).

### **Βασική γραμμή:**

Η βασική γραμμή είναι ένα ορόσημο στην ανάπτυξη λογισμικού που σημειώνει την ολοκλήρωση μιας φάσης, εργασίας ή κύκλου ανάπτυξης και συνοδεύεται από έναν αριθμό εγκεκριμένων στοιχείων σχηματισμών λογισμικού που πρέπει να παραδοθούν.

Σύμφωνα με τον προηγούμενο ορισμό, μια βασική γραμμή παρέχει το επίσημο πρότυπο πάνω στο οποίο μπορεί να βασιστεί η επακόλουθη εργασία και γι' αυτό τον λόγο οι βασικές γραμμές αποτελούν μέρος του προγραμματισμού της ανάπτυξης του λογισμικού και πρέπει να καθιερώνονται όσο το δυνατόν νωρίτερα στην ανάπτυξη.

## **Σύνοψη ενότητας**

---

Η συναρμολόγηση κάθε εφαρμογής λογισμικού ως ενός σχηματισμού (configuration) επιμέρους συστατικών λογισμικού καθένας εκ των οποίων μπορεί να ακολουθεί τον δικό του κύκλο ζωής και τις δικές του μεταβολές γεννά την ανάγκη της διοίκησης σχηματισμών λογισμικού. Κάθε συστατικό λογισμικού μπορεί να απαντάται σε παράλληλες εκδόσεις οι οποίες ονομάζονται «παραλλαγές» και διαφέρουν μεταξύ τους ως προς κάποια χαρακτηριστικά, όπως η γλώσσα ή το λειτουργικό σύστημα. Κάθε παραλλαγή μπορεί να έχει πολλές εκδόσεις οι οποίες ενσωματώνουν διορθώσεις ή/και επεκτάσεις. Ένα σύνολο εγκεκριμένων συστατικών, τα οποία συγκροτούν μία έκδοση του λογισμικού έτοιμη για διανομή, ονομάζεται «βασική γραμμή».

## ΕΝΟΤΗΤΑ 12.2. ΔΙΟΙΚΗΣΗ ΣΧΗΜΑΤΙΣΜΩΝ ΛΟΓΙΣΜΙΚΟΥ

Εκτός από το πρόβλημα του ορισμού και της κατασκευής των συστατικών στοιχείων λογισμικού, ο κατασκευαστής πρέπει να αντιμετωπίσει το πρόβλημα της διοίκησης (διαχείρισης) των σχηματισμών λογισμικού που προκύπτουν σε μία εφαρμογή. Πρόκειται για μια ιδιαίτερα κρίσιμη εργασία η οποία τρέχει παράλληλα με τις εργασίες ανάπτυξης λογισμικού. Στη συνέχεια δίνουμε τον ορισμό της διοίκησης σχηματισμών λογισμικού, όπως αυτός σημειώνεται στο πρότυπο IEEE std 729-1983.

### **Διοίκηση σχηματισμών λογισμικού:**

Η διοίκηση σχηματισμών λογισμικού είναι η δραστηριότητα του προσδιορισμού και του χαρακτηρισμού των στοιχείων σχηματισμών λογισμικού σε ένα σύστημα, του ελέγχου της αποδέσμευσης και της αλλαγής αυτών των στοιχείων σε όλο τον κύκλο ζωής του συστήματος, της καταγραφής και της έκθεσης της κατάστασης αυτών, καθώς και των αιτήσεων αλλαγών και της επαλήθευσης της πληρότητας και της ορθότητάς τους.

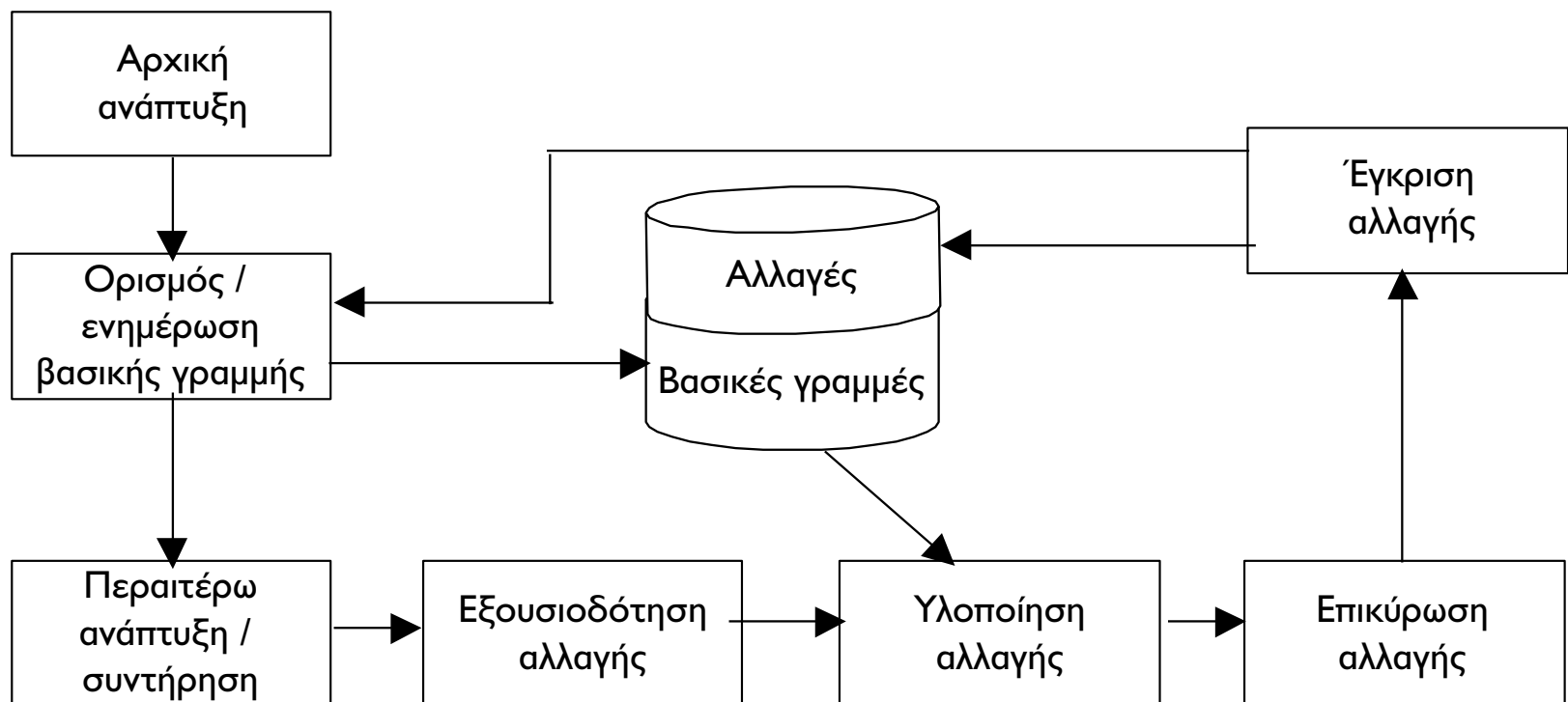
Από τον προηγούμενο ορισμό φαίνονται οι τέσσερις βασικές εργασίες της διοίκησης σχηματισμών λογισμικού, που είναι οι ακόλουθες:

- Καθορισμός σχηματισμών (προσδιορισμός και καθορισμός των στοιχείων σχηματισμών λογισμικού και των βασικών γραμμών).
- Έλεγχος σχηματισμών (έλεγχος των αλλαγών και των τροποποιήσεων των βασικών γραμμών).
- Έλεγχος ποιότητας (επαλήθευση και επικύρωση των αλλαγών στα στοιχεία σχηματισμών λογισμικού).
- Έκθεση κατάστασης σχηματισμών (καταγραφή και έκθεση της κατάστασης των στοιχείων σχηματισμών λογισμικού).

Στο Σχήμα 12.3 παρουσιάζεται μια συνοπτική περιγραφή της δραστηριότητας της διοίκησης σχηματισμών λογισμικού. Όπως φαίνεται στο σχήμα, όλες οι δραστηριότητες της διοίκησης σχηματισμών λογισμικού περιστρέφονται γύρω από τις βασικές γραμμές. Μόλις σταθεροποιηθεί ένα αρχικό επίπεδο στο προϊόν, καθιερώνεται η πρώτη βασική γραμμή. Κάθε διαδοχικό

σύνολο από επικυρωμένες αλλαγές καθιερώνει μια νέα βασική γραμμή που αποτελεί τον θεμέλιο λίθο για περαιτέρω ανάπτυξη.

**Σχήμα Ι2.3** Συνοπτική περιγραφή της δραστηριότητας της διοίκησης σχηματισμών λογισμικού.



Δεν υπάρχουν τυποποιημένοι κανόνες και μέθοδοι για την πραγματοποίηση των εργασιών της διοίκησης σχηματισμών λογισμικού. Οι ανάγκες και οι ειδικές συνθήκες κάθε έργου πρέπει να λαμβάνονται υπόψη, έτσι ώστε η διοίκηση σχηματισμών λογισμικού να μην παρακωλύει την παραγωγικότητα με τον εξαναγκασμό υπερβολικών γραφειοκρατικών διαδικασιών. Από την άλλη, οι εργασίες της διοίκησης σχηματισμών λογισμικού δεν θα πρέπει να είναι τόσο χαλαρές ώστε να επιτρέπουν τη δημιουργία χάους. Μερικά χαρακτηριστικά παραδείγματα τέτοιων ανεπιθύμητων καταστάσεων είναι τα ακόλουθα:

- Δεν είναι καταγεγραμμένες οι μεταβολές που έχουν πραγματοποιηθεί σε κάποιο στοιχείο λογισμικού και τα μέλη της ομάδας ανάπτυξης δεν ξέρουν ποια έκδοσή του πρέπει να χρησιμοποιήσουν.
- Δεν είναι γνωστά τα στοιχεία σχηματισμών λογισμικού που συγκροτούν μια αποδέσμευση η οποία έχει εγκατασταθεί σε πελάτη, με αποτέλεσμα να μην είναι δυνατός ο εντοπισμός της πηγής ενός προβλήματος που αναφέρει ο πελάτης.
- Εκδόσεις ενός στοιχείου που στην πραγματικότητα ήταν παραλλαγές και έπρεπε να συνυπάρχουν με τις υπόλοιπες θεωρήθηκαν διορθώσεις και τις αντικατέστησαν, με αποτέλεσμα να μην είναι πλέον διαθέσιμες για συντήρηση – επέκταση.

Γίνεται, λοιπόν, αντιληπτό ότι η διοίκηση σχηματισμών λογισμικού είναι μια κρίσιμης σημασίας εργασία η οποία, αν και δεν σχετίζεται με αυτό καθαυτό το αντικείμενο της κατασκευής λογισμικού, είναι αναγκαία, ιδιαίτερα στους μεγάλους οργανισμούς ανάπτυξης και στις μεγάλες εφαρμογές λογισμικού.

## ΕΝΟΤΗΤΑ 12.3. ΕΡΓΑΣΙΕΣ ΔΙΟΙΚΗΣΗΣ ΣΧΗΜΑΤΙΣΜΩΝ ΛΟΓΙΣΜΙΚΟΥ

Στην ενότητα αυτή θα ασχοληθούμε με την περιγραφή των τεσσάρων βασικών εργασιών της διοίκησης σχηματισμών λογισμικού τις οποίες εντοπίσαμε στην Ενότητα 12.2.

### 12.3.1. Καθορισμός σχηματισμών

Η δραστηριότητα καθορισμού σχηματισμών περιλαμβάνει τις ακόλουθες ενέργειες:

1. Καθορισμός των απαραίτητων αντικειμένων που πρέπει να τεθούν κάτω από έλεγχο (δηλαδή ποια θα είναι τα στοιχεία σχηματισμών λογισμικού).
2. Καθορισμός των ονομάτων τους.
3. Καθορισμός των μεταξύ τους σχέσεων.
4. Καθορισμός των βασικών γραμμών.

Συστατικά στοιχεία λογισμικού που πρέπει να είναι κάτω από έλεγχο είναι συνήθως τα ακόλουθα:

- Απαιτήσεις του πελάτη
- Προδιαγραφές
- Τεκμηρίωση ανάλυσης
- Τεκμηρίωση σχεδίασης
- Πηγαίος κώδικας
- Κώδικας σε μορφή object
- Πλάνα ελέγχου
- Περιπτώσεις ελέγχου
- Εκθέσεις ελέγχου
- Εργαλεία ανάπτυξης
- Εγχειρίδια συντήρησης
- Εγχειρίδια χρηστών

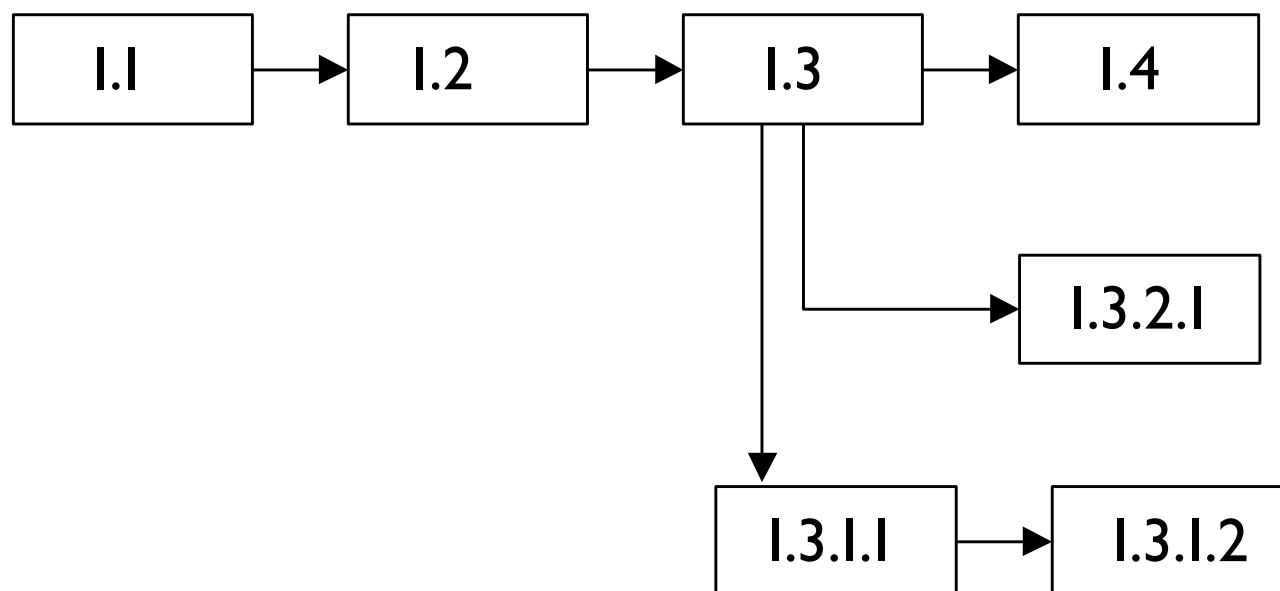
Η διοίκηση των στοιχείων σχηματισμών λογισμικού προϋποθέτει ότι κάθε στοιχείο έχει ξεχωριστό όνομα. Το όνομα ενός στοιχείου σχηματισμών λογισμικού έχει δύο μέρη: το πρώτο μέρος προσδιορίζει το ίδιο το στοιχείο ως οντότητα και το δεύτερο μέρος προσδιορίζει την έκδοσή του. Το όνομα του πρώτου μέρους συνήθως κατασκευάζεται από ένα ιεραρχικό σχήμα ονοματολογίας, όπου κάθε όνομα αποτελείται από έναν αριθμό συστατικών. Κάθε τέτοιο συστατικό αντιστοιχεί σε μια πιο λεπτομερή άποψη του όλου συστήματος, ξεκινώντας από το όνομα του έργου.

Για παράδειγμα, το όνομα `XPR_TOOLS_IMAGE_GIF` μπορεί να προσδιορίζει τον κώδικα διαχείρισης μιας εικόνας `GIF`, ο οποίος κώδικας μπορεί να είναι μέρος της μονάδας `IMAGE`, η οποία με τη σειρά της είναι μέρος του υποσυστήματος `TOOLS` του έργου `XPR`. Η περιγραφή αυτής της ιεραρχίας και γενικότερα της φιλοσοφίας ονοματολογίας είναι ένα κρίσιμο έγγραφο του έργου, διότι επιτρέπει τη δημιουργία κατανοητών και ανιχνεύσιμων ονομάτων οποιουδήποτε στοιχείου σχηματισμών λογισμικού.

Για να μπορούμε να χειριστούμε διαφορετικές εκδόσεις του συστήματος πρέπει κάθε έκδοση ενός στοιχείου σχηματισμών λογισμικού να έχει μοναδικό όνομα. Ο προσδιοριστής μιας έκδοσης είναι ένα ζευγάρι της μορφής `{variation_id, revision_id}` όπου `variation_id` είναι ο προσδιοριστής της παραλλαγής και `revision_id` είναι ο προσδιοριστής αναθεώρησης. Κάθε προσδιοριστής μπορεί να είναι τόσο απλός όσο ένας αριθμός ή τόσο πολύπλοκος όσο ένα σύνολο από ονόματα-σημαίες (`flags, switches`) που δηλώνουν ότι έχουν εφαρμοστεί στο σύστημα κάποιοι συγκεκριμένοι τύποι λειτουργικών αλλαγών (π.χ. `optimized=yes, released=no`).

Ο πιο διαδεδομένος τρόπος είναι αυτός που χρησιμοποιεί ένα σχήμα αριθμών της μορφής «`n.k`», όπου το «`n`» χρησιμοποιείται για να δηλώσει την παραλλαγή και το «`k`» για να δηλώσει την αναθεώρηση (Σχήμα 12.4). Εάν υπάρχουν πολλαπλές παραλλαγές που ξεκινάνε από έναν συγκεκριμένο κόμβο, τότε χρησιμοποιείται ένα νέο ζευγάρι `n.k`. Για παράδειγμα, στο Σχήμα 12.4, ο κόμβος `I.3` σημαίνει την τρίτη αναθεώρηση της πρώτης παραλλαγής, ενώ ο κόμβος `I.3.2.I` σημαίνει την πρώτη αναθεώρηση της δεύτερης παραλλαγής του κόμβου `I.3`.

**Σχήμα Ι2.4** Παράδειγμα ενός δένδρου εκδόσεων.





Η απλότητα του προηγούμενου σχήματος το κάνει ιδιαίτερα ελκυστικό και έτσι υποστηρίζεται από πολλά εργαλεία διαχείρισης εκδόσεων. Επειδή το σχήμα αυτό βασίζεται στη θεώρηση ότι όλες οι εκδόσεις δημιουργούνται ως μια γραμμική σειρά, πράγμα που δεν είναι πάντοτε αλήθεια για τις παραλλαγές, μπορεί να εφαρμοστεί ένα εναλλακτικό σχήμα, το οποίο συνδυάζει πληροφορία με αριθμούς και με συμβολικά ονόματα.

Στην εργασία καθορισμού σχηματισμών ακολουθεί η εξέταση των σχέσεων μεταξύ των στοιχείων σχηματισμών λογισμικού. Οι σχέσεις αυτές είναι πολύ κρίσιμες στην ανάλυση των επιπτώσεων των αλλαγών και γι' αυτόν τον λόγο πρέπει να προσδιορίζονται. Οι σχέσεις μεταξύ των στοιχείων σχηματισμών λογισμικού μπορεί να οριστούν ως υποπεριπτώσεις (υποκλάσεις) της γενικής σχέσης «εξαρτάται\_από». Τέτοιες είναι η σχέση «αποτελείται\_από» μεταξύ των σύνθετων στοιχείων και των μερών τους και η σχέση «παράγει» μεταξύ στοιχείων που αποτελούν είσοδο σε εργαλεία και των στοιχείων που παράγονται από τα εργαλεία αυτά (όπως είναι, για παράδειγμα, τα στοιχεία πηγαίου κώδικα και τα αντίστοιχα στοιχεία κώδικα σε μορφή object που παράγει ο μεταγλωττιστής της γλώσσας προγραμματισμού που χρησιμοποιείται).

Σε μερικές περιπτώσεις η δημιουργία τέτοιων σχέσεων μπορεί να γίνει αυτόματα. Για παράδειγμα, στη γλώσσα προγραμματισμού C ή C++, το περιεχόμενο της μετάφρασης των μονάδων καθορίζεται από τις δηλώσεις τύπου `#include`. Ένα εργαλείο, με τη βοήθεια αυτών των δηλώσεων, θα μπορούσε να κατασκευάσει σχέσεις εξάρτησης αυτού του είδους.

Τέλος, καθορίζονται οι βασικές γραμμές. Αν και οι βασικές γραμμές μπορεί να καθοριστούν σε οποιοδήποτε επίπεδο λεπτομέρειας, οι πιο κοινές είναι οι παρακάτω:

- Βασική γραμμή απαιτήσεων (Requirements Baseline): Αυτή καθιερώνεται μόλις οι απαιτήσεις ολοκληρωθούν και αρχικώς εγκριθούν. Περιλαμβάνεται η λειτουργική ιδέα και ένα ευρετήριο για κάθε επιθυμητή λειτουργία.
- Βασικές γραμμές προδιαγραφών (Specifications Baselines): Αυτές περιλαμβάνουν τις προδιαγραφές του λογισμικού, καθώς και διασταυρώσεις με τις απαιτήσεις.

- Βασικές γραμμές σχεδίασης (Design Baselines): Αυτές καθιερώνονται μόλις το σχέδιο ολοκληρωθεί αρχικώς και εγκριθεί. Τα στοιχεία που περιλαμβάνει είναι κάθε πληροφορία που αφορά αποφάσεις-κλειδιά της σχεδίασης και σχέσεις μεταξύ τους, όπως το αρχιτεκτονικό σχέδιο, το σχέδιο μονάδων, το σχέδιο διεπαφής, το σχέδιο των οντοτήτων δεδομένων και την ιεραρχία των αντικειμένων αν χρησιμοποιούνται αντικειμενοστρεφείς τεχνικές.
- Βασικές γραμμές μονάδων (Component Baselines): Μια βασική γραμμή καθιερώνεται για κάθε μονάδα λογισμικού που κατασκευάζεται, ελέγχεται και εγκρίνεται. Αυτή η βασική γραμμή μπορεί να περιλαμβάνει αμφότερα τον πηγαίο κώδικα και τον κώδικα σε μορφή object.
- Βασική γραμμή συνένωσης (Integration Baseline): Αυτή η βασική γραμμή καθιερώνεται μόλις το σύστημα ή τα σύνθετα συστατικά του κατασκευαστούν.
- Βασική γραμμή ελέγχου (Testing Baseline): Κάθε έλεγχος είναι μια σημαντική επένδυση και γι' αυτόν τον λόγο τα δεδομένα του πρέπει να αποθηκεύονται για μελλοντική αναφορά. Μια τροποποίηση στα αποτελέσματα των ελέγχων μπορεί να φανερώσει ενδεχόμενα προβλήματα τα οποία μπορεί να μην ήταν δυνατό να προσδιοριστούν με άλλον τρόπο.
- Βασική γραμμή λειτουργίας (Operational Baseline): Αυτή η βασική γραμμή καθιερώνεται μόλις το σύστημα παραδοθεί και αρχίσει να λειτουργεί ώστε να αποτελεί τη βάση για περαιτέρω συντήρηση ή ανάπτυξη.
- Βασική γραμμή εργαλείων: Αν και είναι σπανίως σημαντικό, τα εργαλεία που χρησιμοποιούνται στην επεξεργασία στοιχείων σχηματισμών λογισμικού μπορεί να αποτελέσουν πηγή προβλημάτων. Γι' αυτόν τον λόγο, μια βασική γραμμή εργαλείων βοηθά να απομονώνονται προβλήματα που δεν δημιουργούνται από το λογισμικό αυτό καθαυτό.

Μόλις καθιερωθεί μια βασική γραμμή, όλα τα στοιχεία σχηματισμών λογισμικού που τη συνοδεύουν τοποθετούνται στη βάση δεδομένων του

έργου (project database), η οποία επίσης καλείται και βιβλιοθήκη του έργου (project library) ή και αποθήκη λογισμικού (software repository). Ανάλογα με τη φιλοσοφία ανάπτυξης λογισμικού, τα στοιχεία της αποθήκης λογισμικού μπορούν να επαναχρησιμοποιούνται.

Οι βασικές γραμμές πρέπει να προστατεύονται από μη εξουσιοδοτημένες αλλαγές όμως, ταυτόχρονα, πρέπει να επιτρέπεται στους προγραμματιστές να τροποποιούν και να ελέγχουν τον κώδικά τους. Αυτή η ευλυγισία παρέχεται δίνοντας στους προγραμματιστές, όταν το ζητήσουν, λειτουργικά αντίγραφα οποιουδήποτε μέρους της βασικής γραμμής. Έτσι, μπορούν να δοκιμάζουν οποιαδήποτε αλλαγή χωρίς να έρχονται σε σύγκρουση με την εργασία οποιουδήποτε άλλου. Όταν η εργασία τους είναι έτοιμη να εισαχθεί σε μια νέα βασική γραμμή, η δραστηριότητα διαχείρισης αλλαγών εξασφαλίζει ότι κάθε αλλαγή που εισάγεται είναι συμβατή με κάθε άλλη και έτσι διατηρείται η ακεραιότητα του συστήματος.

Κάθε προτεινόμενη μεταβολή πρέπει να ελεγχθεί σε μια δοκιμαστική έκδοση της νέας βασικής γραμμής. Δεν πρέπει να ελέγχονται όλες οι μεταβολές σε ένα μεγάλο βήμα, μια και αυτό θα μπορούσε να αποτρέψει την απομόνωση πιθανών προβλημάτων. Με δεδομένα ότι: α) οι αλλαγές γίνονται σταδιακά στις μονάδες που θα αποτελούν τη δοκιμαστική έκδοση μιας βασικής γραμμής και β) ένας πλήρης και εκτεταμένος έλεγχος είναι πολύ δαπανηρός και δεν είναι δικαιολογημένος για την έγκριση κάθε μεμονωμένης αλλαγής, τα παρακάτω δύο βήματα χρησιμοποιούνται για την τροποποίηση μιας δοκιμαστικής βασικής γραμμής:

1. Προτού γίνει οποιαδήποτε προσπάθεια για την τροποποίηση μιας μονάδας, η μονάδα αυτή «κλειδώνεται», έτσι ώστε να μην μπορούν να γίνουν περαιτέρω αλλαγές πριν από τον έλεγχο της τρέχουσας. Αυτό αποτρέπει την ενσωμάτωση συγκρουόμενων αλλαγών. Το κλείδωμα παύει να ισχύει μετά την ολοκλήρωση των κατάλληλων ελέγχων που επαληθεύουν την προτεινόμενη αλλαγή.
2. Ένας περιοδικός ολοκληρωμένος έλεγχος γίνεται στη δοκιμαστική βασική γραμμή, ώστε να εξασφαλιστεί ότι οι αλλαγές που έγιναν δεν έχουν προκαλέσει κάποια προβλήματα όπως, για παράδειγμα, την επανεμφάνιση

διορθωμένων λαθών. Τέτοιου είδους έλεγχοι αποτελούνται από έναν μεγάλο αριθμό προηγούμενων λειτουργικών περιπτώσεων ελέγχου και έτσι οποιοδήποτε λάθος φανερώνει ότι η νέα αλλαγή προκαλεί προβλήματα. Αν και αυτοί οι έλεγχοι δεν κοστίζουν όσο ο τελικός που καθιερώνει τη νέα βασική γραμμή, βοηθούν να αποκαλυφθούν λάθη νωρίς στην προσπάθεια ενημέρωσης μιας βασικής γραμμής.

## **Δραστηριότητα 2/Κεφάλαιο 12**

Με βάση την απάντηση που δώσατε στη Δραστηριότητα 1, προτείνετε μια ονοματολογία για τα στοιχεία σχηματισμού λογισμικού που ορίσατε (απλά και σύνθετα). Προσπαθήστε να ακολουθήσετε τις κατευθύνσεις που δίνονται στην Ενότητα 12.3.1.

## **Δραστηριότητα 3/Κεφάλαιο 12**

Στην ενότητα αυτή φάνηκε ότι οι παραλλαγές είναι εκδόσεις που γενικά δημιουργούν προβλήματα στον διαχειριστή σχηματισμών λογισμικού. Ένα τέτοιο πρόβλημα είναι και το ότι δεν μπορεί να θεωρηθεί ότι δημιουργούνται ακολουθώντας πάντοτε μια γραμμική σειρά όπως παρουσιάζει το Σχήμα 12.4. Προσπαθήστε να αναφέρετε ένα παράδειγμα τέτοιων εκδόσεων.

Μπορείτε να θεωρήσετε εκδόσεις ενός στοιχείου σχηματισμών το οποίο χρησιμοποιείται για την εμφάνιση ψηφιακών εικόνων και έχει το γενικό προσδιοριστικό «ImageToolkit». Προσπαθήστε να ονομάσετε εκδόσεις του στοιχείου αυτού, οι οποίες τρέχουν κάτω από διαφορετικές πλατφόρμες υλικού και λειτουργικών συστημάτων.

## Δραστηριότητα 4/Κεφάλαιο Ι2

Ένα άλλο σημείο που αξίζει να προσέξουμε είναι και η σημασία των σχέσεων μεταξύ των στοιχείων σχηματισμών λογισμικού στην ανάλυση των επιπτώσεων των αλλαγών. Όσο περισσότερες σχέσεις μεταξύ των στοιχείων σχηματισμών λογισμικού μπορούμε να αποκαλύπτουμε και να καταγράφουμε, και μάλιστα αυτόματα, τόσο καλύτερη ανίχνευση των επιπτώσεων μιας αλλαγής μπορούμε να κάνουμε. Ειδικότερα τέτοιες σχέσεις που προκύπτουν από τα στοιχεία που αποτελούν είσοδο/έξοδο σε εργαλεία μπορούν εύκολα να καταγράφονται αυτόματα.

Προς αυτήν την κατεύθυνση, αναφέρετε ένα παράδειγμα σχέσεως του τύπου «παράγει» μεταξύ ενός στοιχείου πηγαίου κώδικα και του αντίστοιχου παραγομένου στοιχείου με βάση μια γλώσσα προγραμματισμού που γνωρίζετε.

### 12.3.2.Ελεγχος μεταβολών σχηματισμών

Η αναγκαιότητα για αλλαγή είναι άρρηκτα συνδεδεμένη με τη διαδικασία ανάπτυξης και είναι αποτέλεσμα, εκτός οτιδήποτε άλλου, του γεγονότος ότι όσο περνά ο καιρός συλλέγεται περισσότερη γνώση από αμφότερους τους κατασκευαστές λογισμικού και τους πελάτες. Η δραστηριότητα ελέγχου σχηματισμών ασχολείται με τις ενέργειες που πρέπει να γίνονται στην επεξεργασία μεταβολών πάνω σε καθιερωμένες βασικές γραμμές και εξασφαλίζει ότι αυτές επηρεάζουν τον σχηματισμό του λογισμικού με τρόπο προβλέψιμο.

Το πρώτο βήμα στην επεξεργασία μιας μεταβολής είναι η συμπλήρωση μιας φόρμας αίτησης αλλαγής (Change Request Form). Αυτό είναι ένα τυπικό έγγραφο όπου το ενδιαφερόμενο μέρος σημειώνει τη σχετική πληροφορία, όπως η επίδραση της προτεινόμενης αλλαγής σε άλλα συστατικά του συστήματος, το κόστος της αλλαγής, κ.λπ. Ο καθορισμός της μορφής της



φόρμας αυτής γίνεται νωρίς, πριν ακόμα ξεκινήσει η διαδικασία ανάπτυξης λογισμικού και μπορεί να χαρακτηρίζει ολόκληρο τον οργανισμό ανάπτυξης λογισμικού, αποτελώντας το πλάνο της διοίκησης σχηματισμών λογισμικού. Το Σχήμα 12.5 δείχνει μερικά από τα πεδία που μπορεί να περιέχει μια τέτοια φόρμα. Σε μια τυπική διαδικασία ενός μεγάλου κατασκευαστή λογισμικού, η φόρμα αίτησης αλλαγής αποστέλλεται στην Επιτροπή Ελέγχου Αλλαγών (Change Control Board), η οποία αποφασίζει για το εάν θα πρέπει να γίνει δεκτή ή όχι η αλλαγή, αξιολογώντας τις προτεινόμενες αλλαγές ως προς την ουσία του τεχνικού μέρους, τις πιθανές παρενέργειες, τη συνολική επίπτωση σε άλλα στοιχεία σχηματισμών λογισμικού και το προβλεπόμενο κόστος τους. Αν η αλλαγή εγκριθεί, τα αντίστοιχα στοιχεία σχηματισμών λογισμικού «κλειδώνονται» προτού γίνει οποιαδήποτε τροποποίηση (check out) Το κλείδωμα παύει να ισχύει (check in) αφού οι ενημερωμένες εκδόσεις τους έχουν ελεγχθεί και η σωστή λειτουργία τους έχει επιβεβαιωθεί. Ο μηχανισμός ελέγχου δεν παρέχει μόνο ένα μέσο για τον συγχρονισμό μεταξύ των μελών της ομάδας που επιθυμούν να προσπελάσουν μια μονάδα, αλλά παρέχει επίσης και έναν τρόπο ελέγχου προσπέλασης πάνω στα στοιχεία σχηματισμών λογισμικού.

**Σχήμα I2.5** Παράδειγμα μιας φόρμας αίτησης αλλαγών.

## **ΦΟΡΜΑ ΑΙΤΗΣΗΣ ΑΛΛΑΓΗΣ**

Αιτών

Ημερομηνία

Στοιχεία αλλαγής

Αριθμός αλλαγής

Περιγραφή αλλαγής

Προσέγγιση αλλαγής

Προτεραιότητα αλλαγής

Στοιχεία υλοποίησης

Αρμοδιότητα υλοποίησης

Τύπος αλλαγής

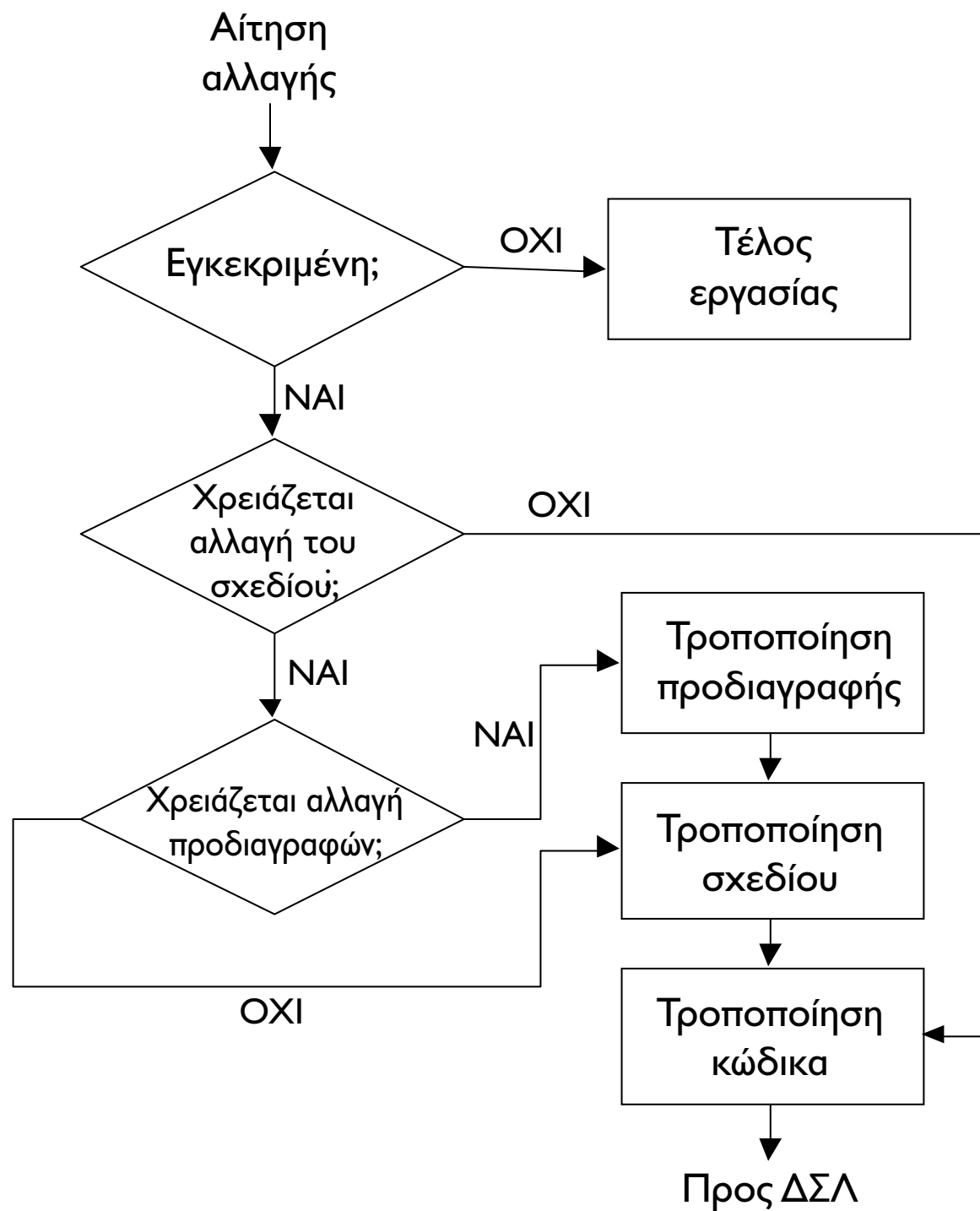
Έκταση αλλαγής

Κόστος αλλαγής

Μονάδες που επηρεάζονται

Συσχετιζόμενες μονάδες λογισμικού

**Σχήμα Ι2.6** Δραστηριότητα ελέγχου αλλαγών.





Η διαδικασία της φόρμας αίτησης/έγκρισης μπορεί να παρακαμφθεί μόνο σε εξαιρετικές καταστάσεις, όπου η αλλαγή διορθώνει ένα καταστροφικό λάθος που προκαλεί την πτώση ολόκληρου του συστήματος, και πάλι όμως η αλλαγή πρέπει να γίνει με τη διαδικασία που περιγράφηκε. Μια προτεινόμενη διαδικασία υλοποίησης μεταβολών παρουσιάζεται στο Σχήμα 12.6. Οι μεταβολές που έγιναν στο επίπεδο του κώδικα θα πρέπει να αντανakλώνται σε όλα τα σχετικά στοιχεία σχηματισμών λογισμικού. Μια αποτυχία σε αυτό οδηγεί αναπόφευκτα στο να χαθεί ο έλεγχος των μεταβολών. Αν, για παράδειγμα, μια τροποποίηση στον κώδικα δεν αντανakλάται και στο επίπεδο του σχεδίου, αργά ή γρήγορα (μάλλον γρήγορα) τα έγγραφα της σχεδίασης θα γίνουν ανεπίκαιρα και δεν θα παρέχουν πλέον τις κατευθυντήριες γραμμές που ακολουθούνται από τον υλοποιημένο κώδικα, κάνοντας πολύπλοκο το έργο της συντήρησης.

Όσο τα στοιχεία σχηματισμών λογισμικού αλλάζουν, είναι σημαντικό να καταγράφονται όλες οι αλλαγές που έγιναν σε καθένα ξεχωριστά. Το σύστημα διαχείρισης αλλαγών μπορεί επίσης να καταγράφει και όλες τις αιτήσεις αλλαγών (εγκεκριμένων ή μη) και τις αναφορές προβλημάτων που έχουν δρομολογηθεί.

Γενικά, καλό είναι να καταγράφεται οποιαδήποτε πληροφορία σχετικά με το τι έγινε, από ποιον, πότε, τι παραμένει να γίνει, οι ειδικές συνθήκες και η τρέχουσα κατάσταση. Στατιστικά στοιχεία από τέτοιες καταγραφές μπορεί να αποκαλύψουν πολλά σχετικά με τα στοιχεία σχηματισμών λογισμικού και, τελικά, για την προσέγγιση ανάπτυξης λογισμικού που ακολουθείται από τον κατασκευαστή. Για παράδειγμα, ακανόνιστα σχήματα αλλαγών σε μια μονάδα μπορεί να φανερώσουν ότι υπάρχει κάποιο πρόβλημα στη σχεδίασή της ή ότι ο προγραμματιστής που είναι υπεύθυνος γι' αυτήν τη μονάδα χρειάζεται επιμόρφωση.

Ένα άλλο κρίσιμο θέμα είναι η διαχείριση της αποθήκης λογισμικού. Εκτός από το μηχανισμό κλειδώματος που περιγράφηκε προηγουμένως, για την προσπέλαση στα στοιχεία σχηματισμών λογισμικού υπάρχουν προβλήματα που προκύπτουν από τον μεγάλο αριθμό των εκδόσεων. Η βιβλιοθήκη λογισμικού συχνά μεγαλώνει, περιλαμβάνοντας συστατικά στοιχεία που δεν

είναι πλέον ενεργά, είναι όμως απαραίτητη η αποθήκευση πληροφοριών σχετικά με αυτά.

Δημιουργούνται, λοιπόν, και προβλήματα στην αποθήκευση και την οργάνωση των βιβλιοθηκών λογισμικού. Με σκοπό την αποθήκευση με οικονομικό τρόπο, χρησιμοποιείται η τεχνική των «διαφορών», που σημαίνει ότι αποθηκεύεται πλήρως μόνο μία έκδοση και όλες οι υπόλοιπες αποθηκεύονται υπό τη μορφή διαφορών από αυτή, οι οποίες ονομάζονται «deltas». Ένα delta δεν είναι τίποτε άλλο παρά ένα σύνολο από εντολές (scripts) που μετασχηματίζουν μία έκδοση ενός εγγράφου σε μία άλλη. Είναι αντιληπτό ότι το πρόβλημα γίνεται τόσο πιο πολύπλοκο όσο μεγαλώνει το μέγεθος του έργου λογισμικού και ο χρόνος ζωής του. Επίσης, είναι φανερό ότι η εργασία αυτή είναι πολύ σύνθετη για να γίνεται με το χέρι και εδώ είναι που εργαλεία για τη διοίκηση σχηματισμών λογισμικού γίνονται αναγκαία.

### **12.3.3. Έλεγχος ποιότητας σχηματισμών**

Το αντικείμενο του ελέγχου ποιότητας σχηματισμών είναι η εξασφάλιση ότι κάθε αλλαγή έχει υλοποιηθεί σωστά. Οι έλεγχοι ποιότητας σχηματισμών είναι απαραίτητοι πριν από τα σημεία αναφοράς κάθε φάσης ανάπτυξης, όπως είναι η καθιέρωση των βασικών γραμμών. Επιπλέον, η συχνότητα και το βάθος του ελέγχου ποιότητας θα πρέπει να είναι αυξανόμενα με την πρόοδο της διαδικασίας ανάπτυξης.

Ο έλεγχος ποιότητας σχηματισμών πρέπει να γίνεται σε περιοδική βάση για να εξασφαλιστεί ότι ακολουθούνται σωστά οι πρακτικές και οι δραστηριότητες της διοίκησης σχηματισμών λογισμικού. Ένας τέτοιος έλεγχος μπορεί να γίνει είτε απευθείας από το προσωπικό της διοίκησης σχηματισμών λογισμικού είτε από κάποια άλλη ανεξάρτητη δραστηριότητα εξασφάλισης ποιότητας. Η ομάδα ελέγχου ποιότητας καλό είναι να αποτελείται από προσωπικό με τεχνικά προσόντα που δεν είναι εμπλεκόμενο στα συγκεκριμένα θέματα που ελέγχονται.

Κάθε έλεγχος ποιότητας σχηματισμού θα πρέπει να βασίζεται σε ένα τεκμηριωμένο πλάνο της διοίκησης σχηματισμών λογισμικού και θα πρέπει να απαντά στις ακόλουθες ερωτήσεις:

- Έχουν υλοποιηθεί σωστά οι αλλαγές που αναφέρονται σε κάθε φόρμα αίτησης αλλαγής;
- Έχουν ακολουθηθεί σωστά τα αντίστοιχα πρότυπα της ανάπτυξης λογισμικού;
- Έχουν ακολουθηθεί οι ενέργειες της διοίκησης σχηματισμών λογισμικού για την σημείωση της αλλαγής, την καταγραφής τη και την αναφορά της;
- Έχουν ενημερωθεί κατάλληλα τα συσχετιζόμενα στοιχεία σχηματισμών λογισμικού;

#### **12.3.4. Έκθεση κατάστασης σχηματισμών**

Ο σκοπός της έκθεσης κατάστασης σχηματισμών είναι η συντήρηση μιας συνεχόμενης εγγραφής της κατάστασης όλων των στοιχείων που είναι στις βασικές γραμμές. Η έκθεση κατάστασης παίζει σημαντικό ρόλο στην επιτυχία ενός μεγάλου έργου λογισμικού, αφού βελτιώνει την επικοινωνία μεταξύ όλων των ανθρώπων που εμπλέκονται σε αυτό. Οι ερωτήσεις που απαντώνται στην έκθεση κατάστασης είναι:

- 1) Τι συνέβη στον σχηματισμό;
- 2) Από ποιον;
- 3) Πότε;
- 4) Τι επηρεάζεται από κάθε συμβάν;

Οι απαντήσεις στα ερωτήματα αυτά περιλαμβάνονται στην έκθεση κατάστασης, η οποία απαιτεί τουλάχιστον την καταγραφή των παρακάτω πληροφοριών:

- Πότε καθιερώθηκε κάθε βασική γραμμή.
- Τεκμηρίωση της κατάστασης κάθε βασικής γραμμής.
- Πότε κάθε στοιχείο σχηματισμών λογισμικού και κάθε αλλαγή του περιήλθαν στη βασική γραμμή.
- Μια περιγραφή κάθε στοιχείου σχηματισμών λογισμικού.

- Ένας κατάλογος στοιχείων σχηματισμών λογισμικού που δείχνει κάθε στοιχείο με την ημερομηνία δημιουργίας του, την τρέχουσα έκδοση και τις εκδόσεις των μερών του.
- Μια καταγραφή των αλλαγών όπου να φαίνεται η ιστορία όλων των αλλαγών που έγιναν σε κάθε στοιχείο σχηματισμών λογισμικού.
- Η περιγραφή και η κατάσταση κάθε αλλαγής στο λογισμικό.
- Οι αλλαγές που έχουν σχεδιαστεί για κάθε προκαθορισμένη μελλοντική βασική γραμμή.
- Όλες οι εκθέσεις δυσλειτουργίας.
- Όλες οι αιτήσεις αλλαγών.

### **Σύνοψη ενοτήτων 12.2 - 12.3**

---

*Η διοίκηση σχηματισμών λογισμικού είναι μια εργασία που εκτελείται καθ' όλο τον κύκλο ζωής και για όλα τα παραγόμενα συστατικά λογισμικού. Αναλύεται σε τέσσερις επιμέρους εργασίες, αυτές του καθορισμού σχηματισμών, του ελέγχου μεταβολών σχηματισμών, του ελέγχου ποιότητας, καθώς και της έκθεσης κατάστασης σχηματισμών λογισμικού. Οι εργασίες αυτές διασφαλίζουν το ελέγξιμο των χαρακτηριστικών του λογισμικού που αποδεσμεύεται προς το πραγματικό περιβάλλον χρήσης του (release).*

## ΕΝΟΤΗΤΑ 12.4. ΕΡΓΑΛΕΙΑ ΔΙΟΙΚΗΣΗΣ ΣΧΗΜΑΤΙΣΜΩΝ ΛΟΓΙΣΜΙΚΟΥ

Η εργασία της διοίκησης σχηματισμών λογισμικού μπορεί να βοηθηθεί σημαντικά από τη χρήση εργαλείων. Εκτός από εμπορικά συστήματα, υπάρχει ένας αριθμός από εργαλεία τα οποία διατίθενται δωρεάν και χειρίζονται εργασίες ελέγχου εκδόσεων και κατασκευής του συστήματος. Πολλά από αυτά έχουν ενσωματωθεί σε διανομές (distributions) του λειτουργικού συστήματος Unix, ενώ τελευταία παρατηρείται τάση να δημιουργούνται εκδόσεις για περιβάλλον Windows. Χαρακτηριστικά παραδείγματα τέτοιων εργαλείων είναι τα RCS, SCCS και MAKE, μια σύντομη αναφορά στα οποία ακολουθεί.

### RCS

Το RCS (Revision Control System) σχεδιάστηκε για να υποστηρίξει την παράλληλη εργασία πολλών διαφορετικών ανθρώπων σε πηγαίο κώδικα αλλά και σε τεκμηρίωση λογισμικού και, γενικά, σε οποιοδήποτε κείμενο ή αρχείο υπόκειται σε πολλαπλές τροποποιήσεις κατά τη διάρκεια της ζωής του. Η βασική ιδέα που υλοποιεί το RCS είναι η ελεγχόμενη πρόσβαση στο αρχείο που μεταβάλλεται, προκειμένου να αποφεύγεται το χάος της παράλληλης ενημέρωσης από πολλές πλευρές.

Όταν κάποιος θέλει να κάνει μεταβολές σε κάποιο αρχείο, λόγου χάρη πηγαίου κώδικα, πρώτα το δεσμεύει (το RCS χρησιμοποιεί την ορολογία «check out»), έτσι ώστε κανείς άλλος να μην μπορεί να το μεταβάλει. Με την ολοκλήρωση των μεταβολών, το αρχείο επανέρχεται («check in») στη γραμμή παραγωγής και είναι διαθέσιμο στην ομάδα ανάπτυξης. Με τον τρόπο αυτό, διασφαλίζεται η ελέγξιμη και πάντως όχι άναρχη πραγματοποίηση τροποποιήσεων σε αρχεία πηγαίου κώδικα αλλά και γενικότερα σε προϊόντα τεκμηρίωσης λογισμικού.

### SCCS

Ο σκοπός του SCCS (Source Code Control System) είναι να διευκολύνει τη συντήρηση διαφορετικών εκδόσεων ενός συστήματος με οικονομικό τρόπο αποθήκευσης. Το SCCS καταγράφει πότε έγινε μια αλλαγή σε μία μονάδα



και από ποιον, ενώ ταυτόχρονα εξασφαλίζει ότι μόνο ένας προγραμματιστής τη φορά μπορεί να ενημερώνει οποιαδήποτε μονάδα λογισμικού.

Όταν μια νέα μονάδα προστίθεται στο SCCS, της δίνεται ο αριθμός έκδοσης 1.0 και όλες οι επόμενες αλλαγές καταγράφονται με τη μορφή deltas. Ένα delta δεν περιέχει την νέα μορφή της μονάδας, αλλά μόνο την πληροφορία του τι έχει αλλάξει σε αυτή, δηλαδή ποιες γραμμές έχουν σβηστεί, ποιες τροποποιήθηκαν, ποιες νέες προστέθηκαν κ.λπ. Έχοντας αποθηκευμένα όλα τα deltas μπορούμε, εφαρμόζοντάς τα διαδοχικά στη βάση μιας μονάδας (δηλαδή στην έκδοση 1.0), να πάρουμε οποιαδήποτε έκδοση της μονάδας αυτής. Το μόνο μειονέκτημα αυτής της προσέγγισης είναι ότι τα deltas μπορεί να γίνουν ασύμβατα αν η ανάπτυξη της μονάδας προχωρεί παράλληλα (δηλαδή αν έχει παραλλαγές).

## MAKE

Το MAKE είναι ένα εργαλείο που χειρίζεται τις εξαρτήσεις παραγωγής μεταξύ των μονάδων αρχικού, αντικειμενικού (object) και εκτελέσιμου κώδικα ενός συστήματος. Το MAKE χρησιμοποιεί τις πληροφορίες που αποθηκεύει το λειτουργικό σύστημα σε κάθε αρχείο (ημερομηνία και ώρα δημιουργίας, flag κατάστασης) με σκοπό να διακρίνει ποια από τα αρχεία που περιέχουν αντικειμενικό ή εκτελέσιμο κώδικα δεν είναι σε συμφωνία με τα αντίστοιχα αρχεία που περιέχουν τον αρχικό κώδικα, ώστε να εκτελέσει τις εντολές που θα επαναφέρουν το σύστημα σε ενημερωμένη κατάσταση.

Ο χρήστης πρέπει να πληροφορήσει το MAKE για τις εξαρτήσεις που υπάρχουν μεταξύ των αρχείων δημιουργώντας ένα αρχείο κειμένου. Αυτό το αρχείο, που καλείται

«makefile», περιέχει εντολές που καθορίζουν ποια εργαλεία πρόκειται να χρησιμοποιηθούν στην κατασκευή του συστήματος, δηλαδή με ποιον τρόπο ο πηγαίος κώδικας μετατρέπεται σε αντικειμενικό και ο αντικειμενικός σε εκτελέσιμο. Όταν συμβεί κάποια μεταβολή σε κάποιο συστατικό πηγαίου κώδικα, το MAKE ενεργοποιεί μόνο τις εντολές που αφορούν την παραγωγή των συστατικών αντικειμενικού κώδικα που εξαρτώνται από αυτό, χωρίς να πραγματοποιεί μεταγλώττιση ολόκληρου του συστήματος (η οποία μπορεί να απαιτεί χρόνο).

Δυστυχώς, ακόμα και συστήματα μετρίου μεγέθους απαιτούν τη δημιουργία μεγάλων αρχείων `makefiles`, τα οποία είναι δύσκολο να κατανοηθούν και να συντηρηθούν. Το πρόβλημα γίνεται δυσκολότερο όταν η σειρά της μετάφρασης του πηγαίου κώδικα είναι σημαντική. Εξαιτίας αυτών των προβλημάτων, μεταγενέστερα περιβάλλοντα ανάπτυξης ενσωματώνουν γεννήτορες `makefiles` που απλοποιούν αυτή την εργασία.

Τα σύγχρονα περιβάλλοντα ανάπτυξης ενσωματώνουν εργασίες διοίκησης σχηματισμών λογισμικού χωρίς να απαιτούν κάποιο ιδιαίτερο εργαλείο για τον σκοπό αυτό.

## ΕΝΟΤΗΤΑ 12.5. ΑΠΑΝΤΗΣΕΙΣ ΔΡΑΣΤΗΡΙΟΤΗΤΩΝ ΑΥΤΟΑΞΙΟΛΟΓΗΣΗΣ

### Δραστηριότητα I/Κεφάλαιο I2

---

Παραδείγματα σύνθετων στοιχείων λογισμικού είναι:

- Το έγγραφο προδιαγραφών απαιτήσεων.
- Το έγγραφο περιγραφής του σχεδίου.
- Το σύνολο του πηγαίου κώδικα.
- Το μοντέλο περιπτώσεων χρήσης.
- Ένα πακέτο ανάλυσης.

Παραδείγματα απλών στοιχείων λογισμικού είναι:

- Μια οποιαδήποτε συγκεκριμένη λειτουργική απαίτηση.
- Το λεπτομερές σχέδιο μίας συγκεκριμένης μονάδας.
- Ο πηγαίος κώδικας μίας συγκεκριμένης μονάδας.
- Μια περίπτωση χρήσης.
- Μια κλάση ανάλυσης.

Όπως είναι γνωστό, τόσο το έγγραφο προδιαγραφών απαιτήσεων από το λογισμικό όσο και το έγγραφο περιγραφής του σχεδίου λογισμικού ενσωματώνουν όλα τα επιμέρους προϊόντα της ανάλυσης απαιτήσεων και της σχεδίασης. Κατά συνέπεια, αν τα θεωρήσουμε απλά στοιχεία του σχηματισμού λογισμικού, χάνουμε πολύτιμη διαχειριστική πληροφορία, όπως είναι οι αλληλοεξαρτήσεις που έχουν αυτά τα προϊόντα μεταξύ τους. Αυτό θα μας δυσκολέψει πολύ στην ανάλυση των επιπτώσεων μιας αλλαγής, αφού δεν θα μπορούμε να εστιάσουμε σε μικρότερη διαχειριστική μονάδα από αυτή του εγγράφου για να εντοπίσουμε με μεγαλύτερη ακρίβεια τα επιμέρους προϊόντα που επηρεάζονται από την αλλαγή. Ακόμα, θα είμαστε αναγκασμένοι με την παραμικρή αλλαγή να εκδίδουμε και μια νέα έκδοση ολόκληρων των εγγράφων αυτών.

Με πολύ απλά λόγια, ένα ατομικό στοιχείο σχηματισμών λογισμικού είναι ένα συστατικό στοιχείο λογισμικού (κλάση, περίπτωση χρήσης, συνάρτηση κ.ά.). Ένας οποιοσδήποτε περιέκτης τέτοιων στοιχείων (μοντέλο περιπτώσεων



χρήσης, μοντέλο κλάσεων ανάλυσης κ.ά.) είναι ένα σύνθετο στοιχείο σχηματισμών λογισμικού.

## Δραστηριότητα 2/Κεφάλαιο 12

---

Εφαρμόζοντας τα αναφερόμενα στην Ενότητα 12.3.1, έχουμε τα παρακάτω ενδεικτικά ονόματα για τα σύνθετα στοιχεία:

- ΕΠΙΚΟΥΡΟΣ\_ΠροδιαγραφέςΑπαιτήσεων
- ΕΠΙΚΟΥΡΟΣ\_ΠροδιαγραφέςΣχεδίασης
- ΕΠΙΚΟΥΡΟΣ\_ΠηγαίοςΚώδικας

Για τα απλά στοιχεία, μια προσέγγιση ονοματολογίας είναι η παρακάτω:

- ΕΠΙΚΟΥΡΟΣ\_ ΠροδιαγραφέςΑπαιτήσεων  
\_ΛειτουργικέςΑπαιτήσεις\_AI
- ΕΠΙΚΟΥΡΟΣ\_ΠροδιαγραφέςΣχεδίασης  
\_ΛεπτομερέςΣχέδιοΜονάδων\_ExecIII
- ΕΠΙΚΟΥΡΟΣ\_ΠηγαίοςΚώδικας \_ExecIII

Είναι φανερό ότι για την παραπάνω ονοματολογία ακολουθήσαμε ένα ιεραρχικό σχήμα, όπου κάθε συνθετικό του ονόματος παριστάνει και τη θέση του στην ιεραρχία που προκύπτει από τη σχέση «αποτελείται\_από» μεταξύ των σύνθετων ΣΣΛ και των μερών τους.

Μην ανησυχείτε αν τα ονόματα σας φαίνονται μεγάλα σε μέγεθος. Η μόνη εναλλακτική περίπτωση είναι να χρησιμοποιήσετε αναγνωριστικά μικρότερου μεγέθους, τα οποία πάλι θα συγκροτούν ένα όνομα που θα περιέχει την πληροφορία της ιεραρχίας. Ένα τέτοιο παράδειγμα είναι το «EP\_SRS\_FR\_AI», το οποίο με λίγη φαντασία μπορούμε να καταλάβουμε ότι αντιστοιχεί στο «ΕΠΙΚΟΥΡΟΣ\_ ΠροδιαγραφέςΑπαιτήσεων \_ΛειτουργικέςΑπαιτήσεις\_AI». Η εμπειρία έχει δείξει ότι είναι προτιμότερο να έχουμε κατανοητά με την πρώτη, έστω, «μεγάλα» ονόματα, από το να ανατρέχουμε σε πίνακες και αποκωδικοποίηση αναγνωριστικών μικρού μεγέθους, αν και αυτό εξαρτάται τόσο από την ιδιοσυγκρασία της ομάδας ανάπτυξης όσο και από τα εργαλεία που μπορεί να χρησιμοποιούνται για τον σκοπό αυτό.

## Δραστηριότητα 3/Κεφάλαιο 12

Ας πάρουμε την πλατφόρμα υλικού PowerPC. Τότε, για παράδειγμα, η έκδοση 1.2 ενός στοιχείου με όνομα «ImageToolkit», που προορίζεται να τρέχει στην πλατφόρμα αυτή και κάτω από το λειτουργικό σύστημα UNIX, θα μπορούσε να ονομαστεί «PowerPC\_UNIX\_ImageToolkit\_1.2».

Ας υποθέσουμε τώρα ότι η επόμενη έκδοση του στοιχείου αυτού αλλάζει πλατφόρμα υλικού και τρέχει, για παράδειγμα, σε μηχανές Intel, επίσης με το λειτουργικό σύστημα UNIX. Μια τέτοια έκδοση είναι παραλλαγή που όμως δεν προκύπτει ως γραμμική σειρά της προηγούμενης, διότι δεν την αντικαθιστά, αλλά αλλάζει κάποιο από τα χαρακτηριστικά της, εν προκειμένω το λειτουργικό σύστημα. Αυτή θα μπορούσε να ονομαστεί «Intel\_UNIX\_ImageToolkit\_1.1».

Μπορείτε να πειραματιστείτε δημιουργώντας ονομασίες όπως Intel\_Win2k\_ImageToolkit\_1.0 (έκδοση 1.0 του ImageToolkit για περιβάλλον Intel/Windows 2000), Intel\_Win9x\_ImageToolkit\_1.0 (έκδοση 1.0 του ImageToolkit για περιβάλλον Intel/Windows 9x), Intel\_Win9x\_ImageToolkit\_1.1 (αναθεώρηση της προηγούμενης - έκδοση 1.1 του ImageToolkit για περιβάλλον Intel/Windows 9x) κ.ο.κ.

## Δραστηριότητα 4/Κεφάλαιο 12

Στη γλώσσα προγραμματισμού C τα αρχεία πηγαίου κώδικα έχουν προέκταση «.c», ενώ τα αντίστοιχα αρχεία κώδικα σε μορφή object στο λειτουργικό σύστημα UNIX έχουν προέκταση «.o». Αν θεωρήσουμε λοιπόν ως στοιχεία σχηματισμού λογισμικού τα αρχεία τύπου «\*.c», καθώς και τα αντίστοιχα παραγόμενα αρχεία τύπου «\*.o», θα μπορούσαμε να παραστήσουμε μια σχέση που συνδέει τους δύο αυτούς τύπους στοιχείων σχηματισμού λογισμικού με τη σχέση «Αρχείο.c\_παράγει\_Αρχείο.o». Η σχέση αυτή αποτελεί εξειδίκευση της γενικής σχέσης «παράγει».

Δεν είναι καλή ιδέα να υποτιμάτε τον ορισμό ακόμη και τέτοιων προφανών σχέσεων. Σε ένα μεγάλο σύστημα λογισμικού δεν εξυπακούεται ότι υπάρχει μόνο ένας τύπος πηγαίων και ενδιάμεσων αρχείων. Πολύ περισσότερο, ακόμη και μετά την έλευση αρκετού χρόνου από την παραγωγή τους (ή την αποχώρηση

των προγραμματιστών που τα κατασκεύασαν), τα συστατικά μιας βιβλιοθήκης λογισμικού πρέπει να είναι γνωστό πώς παράγονται και πώς συσχετίζονται.

## ΒΙΒΛΙΟΓΡΑΦΙΑ

Pressman, R. S., *Software Engineering-A Practitioners Approach*, McGraw-Hill.

Sommerville, I. *Software Engineering*, London: Addison-Wesley.

## ΔΙΑΣΦΑΛΙΣΗ ΠΟΙΟΤΗΤΑΣ ΛΟΓΙΣΜΙΚΟΥ

Σκοπός του κεφαλαίου είναι να εισάγει τον αναγνώστη στις βασικές έννοιες της διασφάλισης ποιότητας λογισμικού, στα πρότυπα και στις διαδικασίες που ακολουθούνται.

Μετά τη μελέτη του κεφαλαίου αυτού, ο αναγνώστης θα είναι σε θέση:

- να αναφέρει δέκα χαρακτηριστικά ποιότητας λογισμικού,
- να περιγράφει τρεις δραστηριότητες για τη διαχείριση ποιότητας λογισμικού,
- να περιγράφει τη γενική διαδικασία διασφάλισης ποιότητας και πέντε βήματα για την πραγματοποίηση επιθεωρήσεων ποιότητας,
- να αναγνωρίζει δύο γενικές κατηγορίες προτύπων διασφάλισης ποιότητας λογισμικού και τρεις κατηγορίες προτύπων τεκμηρίωσης λογισμικού.

### Έννοιες-κλειδιά

---

- Ποιότητα λογισμικού
- Επιθεώρηση ποιότητας
- Εκθέσεις ποιότητας
- Διασφάλιση ποιότητας
- Πρότυπα διαδικασίας
- Πρότυπα τεκμηρίωσης λογισμικού

### Σύνοψη

---

Η διασφάλιση ποιότητας λογισμικού είναι μια δύσκολη διαδικασία, δεδομένου ότι δεν υπάρχουν μέχρι σήμερα ευρέως χρησιμοποιούμενες ποσοτικές αποτιμήσεις

των χαρακτηριστικών του λογισμικού. Η όλη προσπάθεια επικεντρώνεται στην προτυποποίηση τόσο των εργασιών ανάπτυξης λογισμικού όσο και των συστατικών στοιχείων που παράγονται καθ' όλη την ανάπτυξη. Με τη βοήθεια των προτύπων και με την πραγματοποίηση επιθεωρήσεων ποιότητας, οι οποίες ελέγχουν την εφαρμογή της διαδικασίας και των προτύπων περιγραφής των προϊόντων, επιχειρείται να διασφαλιστεί η ποιότητα του λογισμικού.

Κεντρικό ζητούμενο είναι, ασφαλώς, η ικανοποίηση των απαιτήσεων του χρήστη και η έλλειψη σφαλμάτων. Αν και το δεύτερο μπορεί να διασφαλιστεί με τεχνικές δυναμικού ελέγχου, το πρώτο παραμένει και θα παραμένει για πολύ ακόμη το κεντρικό ζητούμενο της ανάπτυξης λογισμικού. Όπως και με τη διοίκηση σχηματισμών, οι εργασίες διασφάλισης ποιότητας λογισμικού εκτελούνται παράλληλα με τις εργασίες ανάπτυξης και, μάλιστα, αν αυτό είναι δυνατό, από διαφορετικές ομάδες. Μια καλά σχεδιασμένη διαδικασία διασφάλισης ποιότητας λογισμικού συνήθως αποδίδει πάντα το κόστος που επιφέρει.

## Εισαγωγικές παρατηρήσεις

---

Σε όλα τα προϊόντα το ζητούμενο είναι η ποιότητα. Η λειτουργία του ανταγωνισμού συνήθως επιφέρει βελτίωση της ποιότητας, με άλλα λόγια καλύτερα προϊόντα στον τελικό καταναλωτή. Οι προμηθευτές πάσης φύσεως συστημάτων, συσκευών και προϊόντων συναγωνίζονται στην παροχή προϊόντων καλύτερης ποιότητας για τα οποία μάλιστα δίνουν και μακρόχρονες εγγυήσεις, οι οποίες τεκμηριώνονται από την ποιότητα των διαδικασιών παραγωγής, που τελικά αντανακλάται σε ποιότητα των προϊόντων.

Τα παραπάνω ισχύουν για όλα ίσως τα προϊόντα εκτός από το λογισμικό. Το λογισμικό είναι το μόνο προϊόν που πωλείται ως έχει και, μάλιστα, αυτό αναγράφεται ρητά στην άδεια χρήσης την οποία είτε υπογράφει είτε διαβάζει σε κάποια «ψιλά γράμματα» ο αγοραστής. Κάτι αντίστοιχο στην περίπτωση, λόγου χάρη, ενός αυτοκινήτου θα ήταν η διατύπωση ότι «σε περίπτωση ατυχήματος που οφείλεται σε κακοτεχνία στην κατασκευή του οχήματος, ουδεμία ευθύνη φέρει ο κατασκευαστής ούτε για την αποκατάσταση του οχήματος ούτε για τις λοιπές συνέπειες». Και όμως, το λογισμικό πωλείται ευρέως λίγο ως πολύ με τέτοιους όρους, ακόμη και

όταν αναλαμβάνει κρίσιμες αποστολές, γεγονός που είναι ενδεικτικό της δυσκολίας διασφάλισης της ποιότητάς του.

Κάτι τέτοιο ίσως ήταν αναμενόμενο, αν λάβει κανείς υπόψη την ελαστικότητα των επιδεχόμενων ερμηνειών των μεθοδολογιών ανάπτυξης λογισμικού, τη μη εξασφαλισμένη αξιοπιστία του υλικού κ.ά. Η διασφάλιση ποιότητας λογισμικού αποτελεί σήμερα ένα ζητούμενο που μόνο κατά προσέγγιση και σε κάποιο βαθμό μπορεί να επιτευχθεί. Αυτό δεν αποτελεί λόγο μη εφαρμογής των όποιων διαδικασιών επιβάλλει, ίσα ίσα αποτελεί λόγο αντιμετώπισης της πρόκλησης. Σήμερα, μάλιστα, πολλοί μεγάλοι πελάτες απαιτούν διαδικασίες διασφάλισης ποιότητας, προκειμένου να επιλέξουν ανάδοχο κατασκευαστή λογισμικού. Μια σύντομη αναφορά στο θέμα επιχειρούμε στο κεφάλαιο αυτό.

Για τον σκοπό αυτό, παράλληλα με τις εργασίες ανάπτυξης λογισμικού και, μάλιστα, ανεξάρτητα από τη φιλοσοφία και τη συγκεκριμένη μεθοδολογία ανάπτυξης που ακολουθείται, πρέπει να τρέχει η εργασία διοίκησης σχηματισμών λογισμικού (*Software Configuration Management*), μια εισαγωγική αναφορά στην οποία έγινε στο κεφάλαιο 12.



## ΕΝΟΤΗΤΑ 13.1. ΠΟΙΟΤΗΤΑ ΛΟΓΙΣΜΙΚΟΥ

### 13.1.1. Η έννοια του σχηματισμού λογισμικού

Η έννοια της ποιότητας ενός έργου λογισμικού έχει πολλές πλευρές και δεν μπορεί να οριστεί με απλό τρόπο. Η ρευστότητα των πραγμάτων στην Τεχνολογία Λογισμικού, την οποία επισημαίνουμε με κάθε ευκαιρία, καθώς και η μη χειροπιαστή υπόσταση του λογισμικού, επιτρέπουν πολλές ερμηνείες της έννοιας «ποιότητα λογισμικού». Συνήθως, όταν αναφερόμαστε στην αποτίμηση της ποιότητας λογισμικού, εννοούμε το βαθμό επίτευξης των προδιαγραφών ενός έργου λογισμικού και την –κατά το δυνατόν– ανυπαρξία σφαλμάτων. Μπορούμε να ισχυριστούμε γενικά ότι αυτή η προσέγγιση της ποιότητας ισχύει για όλους τους τύπους έργων.

Παρά την όποια ελαστικότητα στον ορισμό της έννοιας «ποιότητα», όλοι συμφωνούν ότι ένα από τα ζητούμενα από την ανάπτυξη μιας εφαρμογής λογισμικού είναι η επίτευξη ενός υψηλού επιπέδου ποιότητας για το τελικό προϊόν. Για τον σκοπό αυτό πρέπει να ληφθούν συγκεκριμένα μέτρα κατά την ανάπτυξη, η ευθύνη για την εφαρμογή και τα αποτελέσματα των οποίων ανήκουν στον διαχειριστή ποιότητας λογισμικού (software quality manager). Το σύνολο αυτών των μέτρων και των σχετιζόμενων εγγράφων, διαδικασιών κ.λπ. αναφέρεται ως διαχείριση ποιότητας λογισμικού.

#### **Διαχείριση ποιότητας λογισμικού:**

Η διαχείριση της ποιότητας περιλαμβάνει τον ορισμό ενός συνόλου κατευθυντήριων γραμμών, διαδικασιών και προτύπων που οδηγούν στη δημιουργία λογισμικού καλής ποιότητας, καθώς και τον έλεγχο ότι το σύνολο αυτών των κατευθυντήριων γραμμών ακολουθείται επιμελώς καθ' όλη τη διάρκεια της ανάπτυξης μιας εφαρμογής λογισμικού.

Ωστόσο, η προσπάθεια ορισμού και εφαρμογής τέτοιων κατευθυντήριων γραμμών στο πεδίο του λογισμικού συναντά κάποιες δυσκολίες οι οποίες οφείλονται στις εγγενείς ιδιαιτερότητες του λογισμικού. Τέτοιες είναι οι ακόλουθες:



- Ο σαφής ορισμός και η πληρότητα των προδιαγραφών του λογισμικού, καθώς και η συμφωνία αυτών με τον πελάτη είναι σημαντικό πρόβλημα, που δεσπόζει στην ανάπτυξη λογισμικού. Αν δεχτούμε ότι η ικανοποίηση των προδιαγραφών είναι το ελάχιστο απαιτούμενο της ποιότητας του λογισμικού, τότε διαπιστώνουμε το παράδοξο ότι από την ίδια τη φύση του λογισμικού αυτό το σημείο αναφοράς δεν μπορεί να χαρακτηριστεί ως ακλόνητο.
- Εκτός από την ικανοποίηση των προδιαγραφών, δεν έχουν οριστεί καλά και σαφή μέτρα για την αποτίμηση της ποιότητας του λογισμικού. Η κοινότητα των κατασκευαστών αρκείται συνήθως στη διατύπωση «επιθυμητών χαρακτηριστικών», τα οποία δεν μπορούν ωστόσο να τεκμηριωθούν ως μετρήσιμα στοιχεία ποιότητας.

Η διαχείριση της ποιότητας λογισμικού δεν αφορά μόνο την επιβεβαίωση ότι το λογισμικό αναπτύσσεται χωρίς σφάλματα και σύμφωνα με τις προδιαγραφές του, αλλά σχετίζεται και με ευρύτερα χαρακτηριστικά του λογισμικού τα οποία μπορούν να γίνουν αντιληπτά ως στοιχεία ποιότητας αυτού (Σχήμα 13.1). Ένα από τα σημαντικότερα στοιχεία του σχεδιασμού διασφάλισης της ποιότητας λογισμικού είναι η επιλογή των αιτούμενων χαρακτηριστικών ποιότητας, καθώς και ο καθορισμός και η ένταξη στον οργανισμό ανάπτυξης λογισμικού του τρόπου με τον οποίο μπορούν αυτά να επιτευχθούν.

**Σχήμα Ι3.Ι** Χαρακτηριστικά της ποιότητας λογισμικού.

Ασφάλεια	Μεταφερσιμότητα
Αξιοπιστία	Ευκολία χρήσης
Ικανότητα ανάκτησης	Αποδοτικότητα
Ελεγχιμότητα	Προσαρμοστικότητα
Ευκολία εκμάθησης	Πολυπλοκότητα

Για τη διαχείριση της ποιότητας ενός έργου ανάπτυξης λογισμικού είναι απαραίτητες τρεις δραστηριότητες:

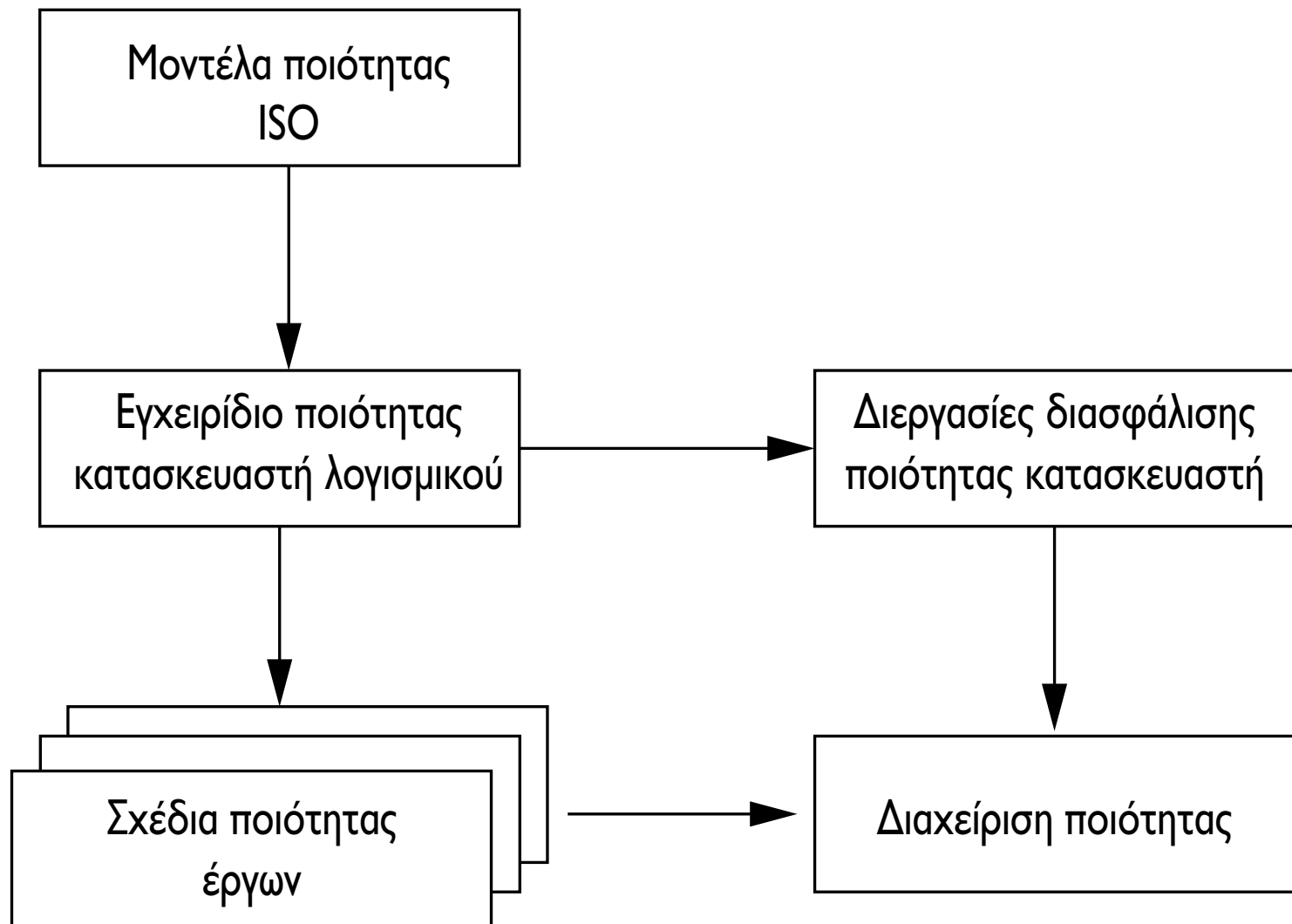
- Η διασφάλιση της ποιότητας που, όπως αναφέραμε και στην αρχή της ενότητας, προϋποθέτει την ένταξη στον οργανισμό ανάπτυξης λογισμικού κατάλληλων εργασιών και προτύπων τα οποία οδηγούν στην κατασκευή λογισμικού υψηλής ποιότητας.
- Ο σχεδιασμός της ποιότητας έργου κατά τον οποίο επιλέγονται τα πιο κατάλληλα πρότυπα για κάθε συγκεκριμένο έργο ανάπτυξης λογισμικού, καθώς και ο τρόπος με τον οποίο μπορούν αυτά να εφαρμοστούν.
- Ο έλεγχος της ποιότητας ο οποίος περιλαμβάνει τον έλεγχο για το αν το υπό κατασκευή λογισμικό έργο αναπτύσσεται σύμφωνα με τα προκαθορισμένα πρότυπα και διαδικασίες. Η εργασία ελέγχου της ποιότητας περιλαμβάνει ένα δικό της σύνολο επιμέρους εργασιών και εγγράφων, τα οποία θα πρέπει να ορίζονται και να εφαρμόζονται κατά τη διαδικασία ανάπτυξης λογισμικού. Οι εργασίες αυτές θα πρέπει να είναι όσο το δυνατόν ξεκάθαρες και εύκολα κατανοητές από εκείνους που αναπτύσσουν το λογισμικό.

Ένα διεθνές πλαίσιο το οποίο μπορεί να χρησιμοποιηθεί για την ανάπτυξη ενός συστήματος διαχείρισης της ποιότητας σε όλες τις βιομηχανίες είναι το ISO-9000, από τον διεθνή οργανισμό προτύπων ISO ([www.iso.ch](http://www.iso.ch)). Το ISO-9000 είναι ένα σύνολο από πρότυπα τα οποία μπορούν να εφαρμοστούν σε ένα ευρύ πεδίο επιχειρήσεων, από κατασκευαστικές έως και επιχειρήσεις παροχής υπηρεσιών. Το ISO-9001 εφαρμόζεται σε οργανισμούς οι οποίοι σχεδιάζουν, αναπτύσσουν και συντηρούν προϊόντα. Περιγράφει διάφορα ζητήματα της εργασίας διασφάλισης της ποιότητας και ορίζει σχετικές διαδικασίες και πρότυπα τα οποία θα πρέπει να ακολουθούνται από έναν τέτοιο οργανισμό.

Το ISO-9001 δεν σχετίζεται με μια συγκεκριμένη βιομηχανία, οπότε τα πρότυπα που ορίζει δεν περιγράφονται με σημαντική λεπτομέρεια. Ωστόσο, ένας οργανισμός, ο οποίος θέλει να αναπτύξει ένα σύστημα διαχείρισης της ποιότητας των προϊόντων του, μπορεί να χρησιμοποιήσει το ISO-9001 για τον ορισμό ενός συνόλου από κατάλληλες εργασίες διασφάλισης της

ποιότητας. Μια εξειδίκευση του ISO-9000 για το λογισμικό αποτελεί το ISO 9000-3. Στο Σχήμα 13.2 εικονίζονται οι σχέσεις μεταξύ του προτύπου ISO-9000, του εγχειριδίου ποιότητας ολόκληρου του οργανισμού, καθώς και των ανεξάρτητων σχεδίων ποιότητας για κάθε έργο.

**Σχήμα Ι3.2** ISO-9000 και διαχείριση ποιότητας.



Ο σχεδιασμός της ποιότητας πρέπει να ξεκινάει παράλληλα με τα αρχικά στάδια της ανάπτυξης μιας εφαρμογής λογισμικού. Ένα σχέδιο ποιότητας εκθέτει τα επιθυμητά χαρακτηριστικά ποιότητας για το παραγόμενο προϊόν τονίζοντας με ιδιαίτερο τρόπο τα σημαντικότερα από αυτά. Επίσης, θα πρέπει να περιγράφει και τον τρόπο με τον οποίο γίνεται η αξιολόγησή τους. Με απλά λόγια, ένα σχέδιο ποιότητας λογισμικού (software quality plan) οριοθετεί τα κριτήρια ποιότητας για μια συγκεκριμένη εφαρμογή λογισμικού.

Η διαχείριση της ποιότητας σε έναν οργανισμό ανάπτυξης λογισμικού θα πρέπει να είναι ανεξάρτητη από αυτή καθεαυτή την ανάπτυξη του λογισμικού, προκειμένου να εξασφαλίζεται ότι η αντιμετώπιση του κρίσιμου ζητήματος της ποιότητας δεν επηρεάζεται από παράγοντες όπως ο προϋπολογισμός ή το χρονοδιάγραμμα του έργου και δεν οδηγείται σε συμβιβασμούς σε βάρος της ποιότητας. Για τον λόγο αυτό είναι επιθυμητή η ανάθεση της διαχείρισης της ποιότητας σε μια ανεξάρτητη ομάδα ατόμων τα οποία βρίσκονται ψηλά στην ιεραρχία του οργανισμού ανάπτυξης λογισμικού. Ενίοτε, και προκειμένου να διασφαλίζεται η επιθυμητή ανεξαρτησία, τον ρόλο αυτό καλούνται να παίξουν εξωτερικοί σύμβουλοι διασφάλισης ποιότητας.

## Σύνοψη ενότητας

---

*Η διαχείριση ποιότητας λογισμικού απαιτεί τον ορισμό ενός συνόλου κατευθυντήριων γραμμών, διαδικασιών και προτύπων για την κατασκευή του λογισμικού, καθώς και τον έλεγχο ότι αυτά εφαρμόζονται καθ' όλη τη διάρκεια της ανάπτυξης μιας εφαρμογής λογισμικού. Οι τρεις γενικές εργασίες που περιλαμβάνονται στη διαχείριση ποιότητας είναι η διασφάλιση, ο σχεδιασμός και ο έλεγχος της ποιότητας του έργου.*

## ΕΝΟΤΗΤΑ 13.2. ΕΠΙΘΕΩΡΗΣΕΙΣ ΠΟΙΟΤΗΤΑΣ ΛΟΓΙΣΜΙΚΟΥ

Η εργασία διασφάλισης της ποιότητας κατά την ανάπτυξη μιας εφαρμογής λογισμικού είναι αρκετά δύσκολη. Αυτό συμβαίνει γιατί είναι δύσκολο να μετρήσει κανείς χαρακτηριστικά του λογισμικού (ακόμα και με τον υποκειμενικό χαρακτήρα πολλών από τις μετρήσεις που σχετίζονται με το λογισμικό) χωρίς κάποια δοκιμαστική χρήση του για εκτεταμένο χρονικό διάστημα. Η βελτίωση της ποιότητας μπορεί να επιτευχθεί με τη μελέτη προϊόντων υψηλής ποιότητας και των μηχανισμών που χρησιμοποιήθηκαν για την κατασκευή τους, με στόχο τη γενίκευσή τους και την ευρύτερη εφαρμογή τους, έτσι ώστε να μπορούν να εφαρμοστούν γενικά στον οργανισμό ανάπτυξης λογισμικού.

Ωστόσο, η σχέση μεταξύ της ίδιας της ανάπτυξης του λογισμικού και της ποιότητας του τελικού προϊόντος είναι πολύπλοκη. Στοιχεία ποιότητας της ίδιας της εργασίας (εξίσου, αν όχι πιο δύσκολα να μετρηθούν) επιδρούν ευθέως στην ποιότητα του παραγόμενου προϊόντος. Η πραγματοποίηση μεταβολών στον τρόπο εκτέλεσης κάποιων εργασιών ανάπτυξης λογισμικού δεν σημαίνει απαραίτητα και βελτίωση στην ποιότητα του τελικού προϊόντος, διότι η ανάπτυξη του λογισμικού είναι μια εργασία περισσότερο επινοητική παρά μηχανική, στην οποία η επίδραση των ατομικών χαρακτηριστικών και της εμπειρίας στο τελικό προϊόν είναι ιδιαίτερα σημαντική. Επιπλέον, εξωτερικοί παράγοντες, όπως η καινοτομία μιας εφαρμογής ή η ανάγκη γρήγορης κυκλοφορίας του προϊόντος εξαιτίας εμπορικών λόγων, μπορεί να έχουν ως αποτέλεσμα χαμηλή ποιότητα ανεξάρτητα τόσο από την προσέγγιση ανάπτυξης λογισμικού όσο και από τη διαδικασία διασφάλισης ποιότητας που ακολουθείται.

Τίποτε από τα παραπάνω δεν υποβιβάζει τη σημασία της εργασίας διασφάλισης της ποιότητας λογισμικού, η οποία παραμένει πολύ μεγάλη. Για την πραγματοποίησή της μπορούμε να διακρίνουμε τα εξής βήματα:

- Ο ορισμός προτύπων για την ανάπτυξη του λογισμικού.  
Παραδείγματα τέτοιων προτύπων είναι ο ορισμός κατευθυντήριων γραμμών για τον τρόπο εφαρμογής της προσέγγισης ανάπτυξης

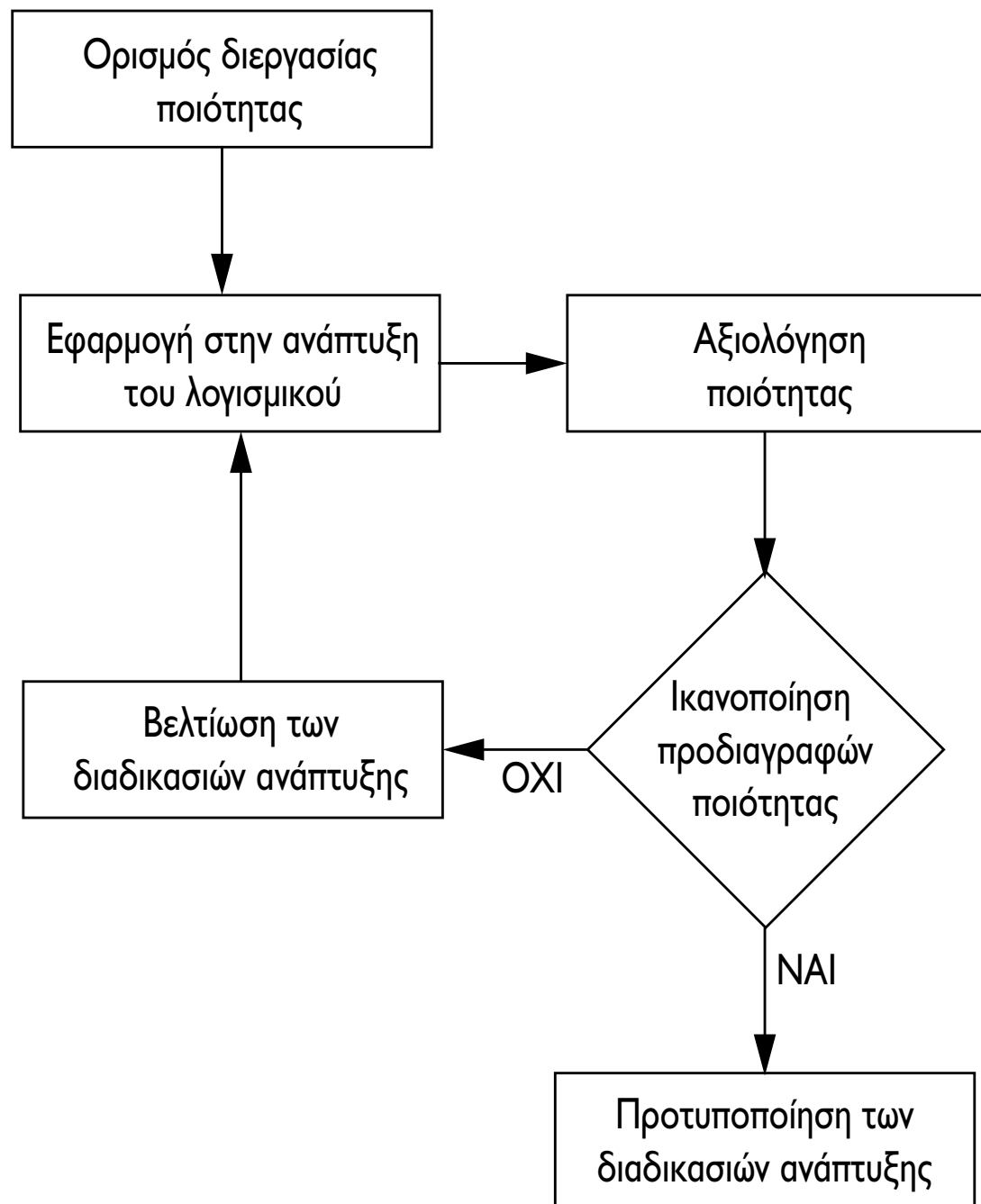
λογισμικού, καθώς και για την αξιολόγηση του τρόπου εφαρμογής του.

- Η εποπτεία (auditing, review) των δραστηριοτήτων ανάπτυξης λογισμικού και η εξασφάλιση ότι τα προκαθορισμένα πρότυπα ακολουθούνται επακριβώς.
- Η αναφορά ποιότητας της εργασίας ανάπτυξης του λογισμικού στον διαχειριστή του έργου και, σε κάποιες περιπτώσεις, στον πελάτη του τελικού προϊόντος.

Ένα αποτέλεσμα που μπορεί να προκύψει από την υιοθέτηση μιας τέτοιας προσέγγισης για τη διασφάλιση της ποιότητας του λογισμικού είναι να αποδειχθεί στην πορεία ότι η προσέγγιση ανάπτυξης λογισμικού είναι ακατάλληλη για τον συγκεκριμένο τύπο του υπό κατασκευή έργου. Στην περίπτωση που κάτι τέτοιο τεκμηριωθεί επαρκώς, τότε θα πρέπει να αναθεωρηθεί συνολικά η ακολουθούμενη προσέγγιση ανάπτυξης λογισμικού, τουλάχιστον για το συγκεκριμένο έργο.



**Σχήμα Ι3.3** Διασφάλιση της ποιότητας βασιζόμενη στη διεργασία ανάπτυξης του λογισμικού.



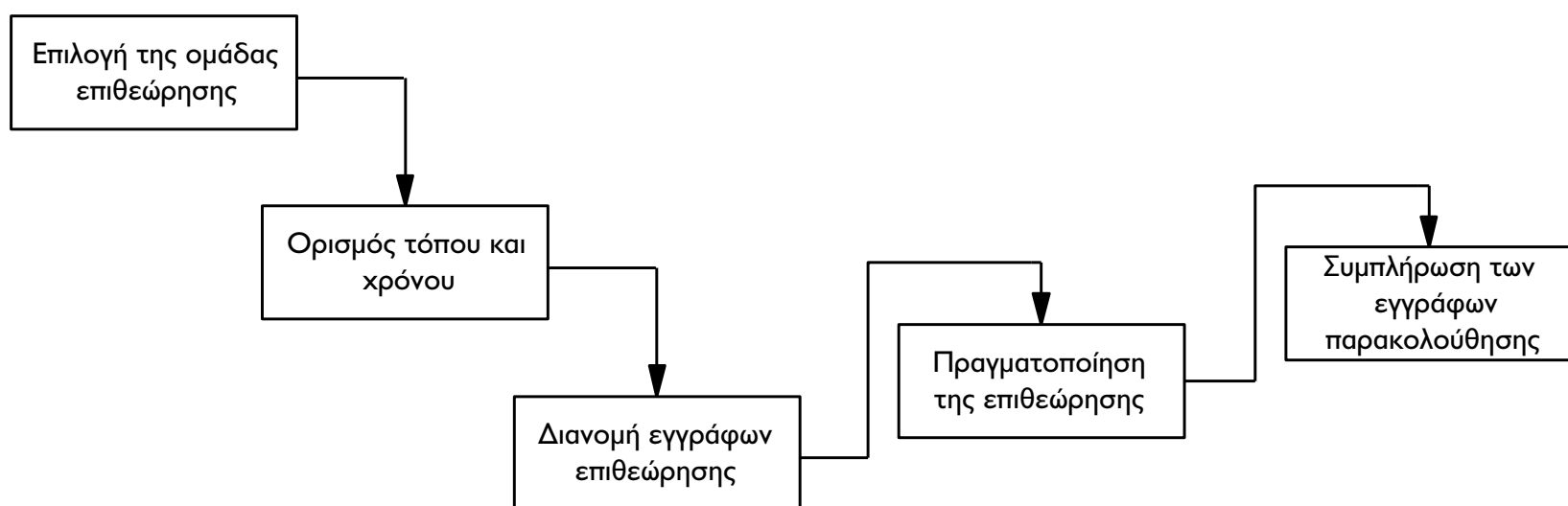
Οι επιθεωρήσεις ποιότητας είναι μια πολύ χρήσιμη μέθοδος για τον έλεγχο της ποιότητας της εργασίας ή/και του προϊόντος αυτής (Σχήμα Ι3.3). Οι επιθεωρήσεις αυτές συνήθως γίνονται από μια μικρή ομάδα ατόμων τα οποία εξετάζουν ένα μέρος ή ολόκληρη τη διεργασία ανάπτυξης του λογισμικού, καθώς και την τεκμηρίωση που τη συνοδεύει, με σκοπό να εντοπίσουν προβλήματα. Τα συμπεράσματα μιας τέτοιας επιθεώρησης καταγράφονται στα έγγραφα παρακολούθησης και παραδίδονται στον εκάστοτε υπεύθυνο για τη διόρθωση των προβλημάτων που αποκαλύπτονται.

### **Επιθεώρηση ποιότητας:**

Η επιθεώρηση της ποιότητας (quality review) περιλαμβάνει μια τεχνική ανάλυση των συστατικών του προϊόντος ή της τεκμηρίωσής του, με σκοπό την αναγνώριση σφαλμάτων ή/και ασυμφωνιών μεταξύ των προδιαγραφών του και του σχεδίου, του κώδικα ή συστατικών της τεκμηρίωσής του λογισμικού.

Η γενική ροή εργασιών για την πραγματοποίηση μιας επιθεώρησης ποιότητας παρουσιάζεται στο Σχήμα Ι3.4. Μια επιθεώρηση ποιότητας διαφέρει από τον έλεγχο του λογισμικού στο γεγονός ότι στην περίπτωση αυτή δεν είναι απαραίτητο να εισέλθει κανείς σε επίπεδο μεγάλης λεπτομέρειας. Κατά την πραγματοποίηση μιας τέτοιας επιθεώρησης δεν είναι απαραίτητη η αναλυτική εξέταση όλων των συστατικών του συστήματος, καθώς ελέγχονται γενικότερα χαρακτηριστικά, όπως η συνεργασία κάποιων από αυτά τα συστατικά, σύμφωνα πάντα με τις απαιτήσεις του πελάτη και τις προδιαγραφές.

**Σχήμα Ι3.4** Η διαδικασία επιθεώρησης ποιότητας.



Η αποστολή της ομάδας που εκτελεί την επιθεώρηση ποιότητας είναι ο εντοπισμός σφαλμάτων και η ενημέρωση των υπευθύνων. Όλα τα συστατικά λογισμικού που παράγονται κατά την ανάπτυξη μπορούν να επιθεωρηθούν. Κατά την ολοκλήρωση μιας επιθεώρησης ποιότητας καταγράφονται οι δραστηριότητες που πραγματοποιήθηκαν. Τα σχόλια και τα συμπεράσματα στα οποία καταλήγει η ομάδα που έκανε την επιθεώρηση καταγράφονται σε ειδικά έντυπα τα οποία ονομάζονται «έγγραφα παρακολούθησης». Τα έγγραφα αυτά ταξινομούνται και εντάσσονται στην τεκμηρίωση του λογισμικού.

Η συχνότητα των επιθεωρήσεων καθορίζεται από τη συνολική εικόνα που δίνει ένα έργο ανάπτυξης λογισμικού: εκεί που παρατηρούνται προβλήματα, είναι πιθανότερο να πρέπει να γίνονται συχνότερες αξιολογήσεις και παρεμβάσεις. Τα αποτελέσματα μιας επιθεώρησης ποιότητας ταξινομούνται σε τρεις κύριες κατηγορίες:

- Καμία ενέργεια: Στην κατηγορία αυτή κατατάσσονται τα έγγραφα παρακολούθησης που περιγράφουν αποκλίσεις ή σφάλματα τα οποία είτε δεν ανταποκρίνονται στην πραγματικότητα είτε το κόστος για τη διόρθωσή τους θεωρείται υπερβολικά υψηλό.
- Αναφορά για διόρθωση: Στην κατηγορία αυτή κατατάσσονται τα έγγραφα που ορίζουν προβλήματα τα οποία είναι απαραίτητο να διορθωθούν. Τα έγγραφα αυτά παραδίδονται στον υπεύθυνο για την επιδιόρθωση των συγκεκριμένων σφαλμάτων.
- Αναθεώρηση του σχεδίου του λογισμικού: Τα έγγραφα που κατατάσσονται στην κατηγορία αυτή ορίζουν ιδιαίτερα σημαντικά σφάλματα για την επιδιόρθωση των οποίων είναι αναγκαία η αλλαγή του σχεδίου του λογισμικού. Πρόκειται, ασφαλώς, για ακραία περίπτωση κατά την οποία, πριν ληφθούν οριστικές αποφάσεις, συνήθως προγραμματίζεται νέα επιθεώρηση.

Μερικά από τα σφάλματα που ανακαλύπτονται κατά τη διάρκεια μιας επιθεώρησης ποιότητας μπορεί να είναι σφάλματα στον ορισμό των προδιαγραφών και των απαιτήσεων του συστήματος. Για τα σφάλματα αυτά θα πρέπει να γίνεται μια εκτίμηση των επιδράσεων που μπορεί να έχει μια πιθανή μεταβολή. Η αλλαγή αυτή ενδεχομένως να συνεπάγεται μια μεγάλης κλίμακας

τροποποίηση της αρχιτεκτονικής του λογισμικού η οποία πρέπει να επιχειρείται να πραγματοποιηθεί με το ελάχιστο δυνατό κόστος.

## ΕΝΟΤΗΤΑ 13.3. ΠΡΟΤΥΠΑ ΛΟΓΙΣΜΙΚΟΥ

Η εργασία της διασφάλισης ποιότητας (και) του λογισμικού είναι συνυφασμένη με την προτυποποίηση της διαδικασίας και των προϊόντων που παράγονται κατά την ανάπτυξη λογισμικού. Η προτυποποίηση είναι, για όλους τους κλάδους της παραγωγής, μια δύσκολη υπόθεση, η οποία γενικά μπορεί να επιτευχθεί σε έναν βαθμό και όχι καθολικά, αποτελεί δε ανταγωνιστικό πλεονέκτημα όσων την επιτυγχάνουν. Στην ενότητα αυτή θα κάνουμε μια σύντομη αναφορά στην προτυποποίηση του λογισμικού και στον ρόλο της στη διασφάλιση ποιότητας.

### 13.3.1. Πρότυπα και διασφάλιση ποιότητας λογισμικού

Μία από τις σημαντικότερες εργασίες της ομάδας που ασχολείται με τη διασφάλιση της ποιότητας είναι ο ορισμός προτύπων λογισμικού. Τα πρότυπα αυτά μπορούν να ταξινομηθούν σε δυο κατηγορίες:

- Τα πρότυπα που χρησιμοποιούνται για την περιγραφή των συστατικών λογισμικού (δομημένα έγγραφα, συμβολισμοί, διαγράμματα, μοντέλα παράστασης λογισμικού κ.ά.).
- Τα πρότυπα που σχετίζονται με την ίδια τη διαδικασία ανάπτυξης του λογισμικού.

Τα πρώτα ορίζουν χαρακτηριστικά τα οποία θα πρέπει να παρουσιάζουν όλα τα συστατικά λογισμικού που ανήκουν σε μία κατηγορία, ενώ τα δεύτερα αφορούν ολόκληρη τη διαδικασία ανάπτυξης του λογισμικού που ακολουθείται στον οργανισμό. Ο ορισμός τέτοιων προτύπων είναι ιδιαίτερα σημαντικός για διαφόρους λόγους, όπως:

- Ενσωματώνουν τις καλύτερες ή, τουλάχιστον, τις πιο κατάλληλες μεθόδους αντιμετώπισης συγκεκριμένων κατηγοριών προβλημάτων. Η εμπειρική αυτή γνώση αποκτάται συνήθως μόνο μετά από

πολλές προσπάθειες, και με την ενσωμάτωσή της σε κάποια πρότυπα αποφεύγεται η επανάληψη παλαιότερων σφαλμάτων.

- Παρέχουν ένα πλαίσιο εργασίας πάνω στο οποίο μπορεί να πραγματοποιηθεί η εργασία διασφάλισης ποιότητας. Δεδομένου ότι (και στο βαθμό που) τα πρότυπα αυτά ενσωματώνουν την καλύτερη δυνατή πρακτική, η διασφάλιση της ποιότητας ισοδυναμεί, πλέον, με μια δραστηριότητα εξασφάλισης ότι τα πρότυπα ακολουθούνται σωστά.
- Βοηθούν στην εξασφάλιση της συνέχειας των εργασιών, δηλαδή στην εξασφάλιση της δυνατότητας ένα άτομο να μπορεί να συνεχίζει εργασίες από το σημείο που τις έχει αφήσει κάποιος άλλος. Τα πρότυπα ενθαρρύνουν την υιοθέτηση των ίδιων μεθόδων από όλους τους εμπλεκόμενους με την ανάπτυξη λογισμικού μέσα σε έναν οργανισμό.

Η δημιουργία τέτοιων προτύπων είναι μια ιδιαίτερα δύσκολη εργασία. Ωστόσο, από μεγάλους διεθνείς φορείς (IEEE, ANSI κ.ά.) έχουν ήδη αναπτυχθεί κάποια διεθνή πρότυπα τα οποία καλύπτουν τη διαδικασία ανάπτυξης λογισμικού, τις γλώσσες προγραμματισμού, καθώς και εργασίες όπως ο ορισμός προδιαγραφών, η διεργασία διασφάλισης της ποιότητας και οι διεργασίες ελέγχου του λογισμικού, στα οποία έχει γίνει αναφορά σε διάφορα σημεία του βιβλίου αυτού. Η ομάδα που ασχολείται με τη διασφάλιση της ποιότητας του λογισμικού θα πρέπει να εξειδικεύει και να καταγράφει τα πρότυπα αυτά σε ένα εγχειρίδιο, για την αποτελεσματικότερη χρησιμοποίησή τους. Το εγχειρίδιο αυτό αποτελεί το εγχειρίδιο διασφάλισης ποιότητας λογισμικού. Στο Σχήμα 13.5 δίνονται κάποια παραδείγματα τέτοιων προτύπων.

### Σχήμα Ι3.5 Πρότυπα προϊόντος και πρότυπα διεργασίας.

Πρότυπα προϊόντος	Πρότυπα διεργασιών
Έντυπα αξιολόγησης	Προδιαγραφή αξιολογήσεων
Ονοματολογία εγγράφων	Διαδικασία καταγραφής ελέγχων ποιότητας
Πρότυπα προγραμματισμού	Διαδικασία κυκλοφορίας του προϊόντος

Οι μηχανικοί λογισμικού θεωρούν αρκετές φορές ότι τα πρότυπα αυτά είναι γραφειοκρατικά και άσχετα με το τεχνικό μέρος της ανάπτυξης λογισμικού. Αυτό συμβαίνει ιδιαίτερα όταν η συμμόρφωση με τα πρότυπα αυτά απαιτεί τη συμπλήρωση μεγάλου αριθμού εντύπων και γενικότερα καταγραφή των εργασιών εργασίας που πραγματοποιούνται. Ωστόσο, όλοι συμφωνούν με τη γενικότερη ανάγκη προτυποποίησης του προϊόντος και των διεργασιών του λογισμικού, καθώς η προσέγγιση αυτή συνεπάγεται γενικά υψηλότερη ποιότητα.

Για την αποτελεσματικότερη καταγραφή και εκμετάλλευση τέτοιων προτύπων, η ομάδα που ασχολείται με τη διασφάλιση ποιότητας του λογισμικού θα πρέπει να ακολουθεί τους παρακάτω κανόνες:

- Στη διαδικασία ορισμού των προτύπων για το προϊόν θα πρέπει να εμπλέκονται και μηχανικοί λογισμικού, οι οποίοι θα κληθούν να τα εφαρμόσουν στην πράξη.
- Είναι αναγκαία η ανά τακτά χρονικά διαστήματα αξιολόγηση και τροποποίηση των προτύπων αυτών, έτσι ώστε να αντικατοπτρίζουν τόσο την εξέλιξη της τεχνολογίας όσο και την τρέχουσα πρακτική. Συνήθως, όταν ορίζονται τα πρότυπα λογισμικού, τοποθετούνται σε ένα εγχειρίδιο και η οποιαδήποτε μετέπειτα αλλαγή τους αποφεύγεται. Ένα τέτοιο εγχειρίδιο είναι απαραίτητο σε μια εταιρεία λογισμικού, όπως απαραίτητη είναι και η εξέλιξή του ανάλογα με τις εκάστοτε συνθήκες και τις εξελίξεις.
- Όταν έχουν οριστεί πρότυπα που σχετίζονται με τον τρόπο εργασίας του προσωπικού για την υλοποίηση του έργου λογισμικού, θα πρέπει να παρέχονται και τα κατάλληλα εργαλεία λογισμικού.

Για την αποφυγή των οποιονδήποτε πιθανών δυσκολιών που μπορεί να παρουσιαστούν κατά τη διαδικασία συμμόρφωσης με ακατάλληλα πρότυπα, ο διαχειριστής της ποιότητας (quality manager) θα πρέπει να αξιολογεί τα προκαθορισμένα πρότυπα και να αποφασίζει ποια από αυτά πρέπει να αποφευχθούν, ποια χρειάζονται αλλαγές και ποια μπορούν να χρησιμοποιηθούν αυτούσια. Τέλος, είναι πιθανή η ανάγκη για τον ορισμό νέων προτύπων, ως συνέπεια κάποιων ιδιαίτερων προδιαγραφών του εκάστοτε έργου



ανάπτυξης λογισμικού, αν και κάτι τέτοιο θα πρέπει να γίνεται μετά από προσεκτική μελέτη.

### 13.3.2. Πρότυπα τεκμηρίωσης λογισμικού

Τα πρότυπα τεκμηρίωσης (documentation standards) αποτελούν τον μόνο τρόπο παράστασης του λογισμικού και της διαδικασίας ανάπτυξής του. Τα έγγραφα τεκμηρίωσης που παράγονται σύμφωνα με πρότυπα έχουν παρόμοια εμφάνιση, δομή και επιδιώκεται η ποιότητά τους να βρίσκεται στα ίδια επίπεδα. Τα έγγραφα αυτά καθιστούν ανιχνεύσιμη κάθε εργασία ανάπτυξης λογισμικού και έτσι θα πρέπει να είναι ακριβή και κατανοητά.

Μπορούμε να διακρίνουμε τρεις διαφορετικούς τύπους προτύπων τεκμηρίωσης:

- Τα πρότυπα της εργασίας τεκμηρίωσης. Τα πρότυπα αυτά ορίζουν τη διαδικασία που πρέπει να ακολουθείται για την παραγωγή των εγγράφων τεκμηρίωσης.
- Τα πρότυπα των εγγράφων τεκμηρίωσης. Τα πρότυπα αυτά ορίζουν γενικά χαρακτηριστικά και δομή των εγγράφων.
- Τα πρότυπα ανταλλαγής εγγράφων τεκμηρίωσης. Κατά τη διεργασία ανάπτυξης του λογισμικού είναι αναγκαία η ανταλλαγή εγγράφων τεκμηρίωσης, συνήθως με ηλεκτρονικό τρόπο (σήμερα επικρατεί το ηλεκτρονικό ταχυδρομείο ή οι εφαρμογές λογισμικού για συντονισμό ομάδων εργασίας), καθώς και η αποθήκευσή τους σε κάποιες βάσεις δεδομένων. Τα πρότυπα συναλλαγής των εγγράφων τεκμηρίωσης περιγράφουν τον τρόπο επικοινωνίας και διακίνησης των εγγράφων, ώστε να αποφεύγεται η αταξία και, τελικά, το χάος.

Τα πρότυπα της εργασίας τεκμηρίωσης ορίζουν τη διαδικασία που πρέπει να ακολουθείται για την παραγωγή των εγγράφων τεκμηρίωσης. Με άλλα λόγια, ορίζουν τις εργασίες που εμπλέκονται στην περιγραφή της δομής των εγγράφων, καθώς και τα εργαλεία λογισμικού που ενδεχομένως να χρησιμοποιούνται για την παραγωγή τέτοιων εγγράφων. Επίσης, τα πρότυπα αυτά θα πρέπει να ορίζουν τις εργασίες ελέγχου και διόρθωσης οι οποίες διασφαλίζουν την υψηλή ποιότητα των εγγράφων τεκμηρίωσης.

Τα πρότυπα διασφάλισης ποιότητας της διεργασίας τεκμηρίωσης θα πρέπει να είναι ευέλικτα, έτσι ώστε να μπορούν να εφαρμόζονται σε όλους τους τύπους εγγράφων. Η δόμηση, ο έλεγχος και η διόρθωση των εγγράφων τεκμηρίωσης σύμφωνα με τα πρότυπα της διεργασίας τεκμηρίωσης είναι μια επαναληπτική διεργασία, η οποία πρέπει να συνεχίζεται μέχρι να επιτευχθεί το ζητούμενο επίπεδο ποιότητας. Το αποδεκτό επίπεδο ποιότητας είναι ανάλογο με τον τύπο του εγγράφου και τους υποψήφιους αναγνώστες του στις εκάστοτε συνθήκες.

Τα πρότυπα των εγγράφων τεκμηρίωσης ισχύουν για όλα τα έγγραφα που παράγονται κατά την ανάπτυξη του λογισμικού. Τα έγγραφα αυτά θα πρέπει να έχουν παρόμοια εμφάνιση, ενώ τα έγγραφα της ίδιας κατηγορίας θα πρέπει να έχουν και παρόμοια δομή. Κάποια παραδείγματα προτύπων για τα έγγραφα τεκμηρίωσης είναι τα εξής:

- Πρότυπα αναγνώρισης των εγγράφων. Τα μεγάλα έργα λογισμικού συνήθως συνοδεύονται από χιλιάδες έγγραφα τεκμηρίωσης. Έτσι, είναι προφανής η ανάγκη για την ύπαρξη μοναδικών αναγνωριστικών για τα έγγραφα αυτά.
- Πρότυπα δόμησης των εγγράφων τεκμηρίωσης. Κάθε κατηγορία εγγράφων που παράγονται κατά την ανάπτυξη λογισμικού πρέπει να ακολουθεί κάποια πρότυπα δόμησης. Τα πρότυπα αυτά ορίζουν τα τμήματα που πρέπει να συμπεριλαμβάνονται στο εκάστοτε έγγραφο όπως και τον τρόπο αρίθμησης των σελίδων, τις επικεφαλίδες των ενοτήτων κ.ά.
- Πρότυπα παρουσίασης των εγγράφων τεκμηρίωσης. Τα πρότυπα αυτά ορίζουν το ύφος της εμφάνισης των εγγράφων τεκμηρίωσης. Εδώ ορίζονται χαρακτηριστικά των εγγράφων όπως οι γραμματοσειρές, τα χρώματα που χρησιμοποιούνται κ.ά.
- Πρότυπα ενημέρωσης των εγγράφων τεκμηρίωσης. Τα πρότυπα αυτά ορίζουν έναν ομοιόμορφο τρόπο ένδειξης των τροποποιήσεων που πραγματοποιούνται σε οποιοδήποτε έγγραφο τεκμηρίωσης.

Τέλος, τα πρότυπα συναλλαγής των εγγράφων τεκμηρίωσης είναι ιδιαίτερα σημαντικά καθώς, συνήθως, κατά τη διάρκεια ανάπτυξης του λογισμικού χρησιμοποιούνται ηλεκτρονικά αντίγραφά τους. Αν υποθέσουμε ότι η χρήση των εργαλείων λογισμικού ορίζεται από τα πρότυπα διεργασίας, τότε τα πρότυπα συναλλαγής ορίζουν τις συμβάσεις για τη χρήση αυτών των εργαλείων.

### **Δραστηριότητα I/Κεφάλαιο I3**

Περιγράψτε με λιγότερες από 100 λέξεις τους λόγους για τους οποίους τα πρότυπα διασφάλισης ποιότητας λογισμικού είναι ιδιαίτερα χρήσιμα.

### **Παράδειγμα I/Κεφάλαιο I3**

Στο Σχήμα I3.6 που ακολουθεί φαίνεται η δομή ενός προτύπου διασφάλισης ποιότητας λογισμικού, το οποίο αφορά ένα συγκεκριμένο έργο. Το πρότυπο κατασκευάζεται ύστερα από απαίτηση του κύριου του έργου, ο οποίος το αναθέτει σε κατασκευαστή λογισμικού. Μπορείτε να παρατηρήσετε ότι το πρότυπο περιλαμβάνει τόσο ζητήματα διοίκησης και οργάνωσης όσο και ζητήματα περιγραφής της διαδικασίας εκτέλεσης και των παραδοτέων του έργου.

## **Σχήμα Ι3.6** Η δομή ενός προτύπου διασφάλισης ποιότητας λογισμικού.

### **0 Ιστορικό αλλαγών εγγράφου**

### **1 Εισαγωγή**

#### 1.1 Σκοπός

#### 1.2 Εμβέλεια

#### 1.3 Διαδικασία αναθεώρησης

#### 1.4 Συντμήσεις – ακρωνύμια

#### 1.5 Λίστα διανομής

#### 1.6 Αναφορές

### **2 Σύντομη περιγραφή του έργου**

### **3 Διοίκηση**

#### 3.1 Οργανόγραμμα ομάδας εκτέλεσης έργου

#### 3.2 Ρόλοι και αρμοδιότητες

#### 3.3 Χρονοδιάγραμμα

#### 3.4 Πρόγραμμα εξασφάλισης ποιότητας

### **4 Τεκμηρίωση παραδοτέων έργου**

#### 4.1 Σκοπός

#### 4.2 Ελάχιστες απαιτήσεις

#### 4.3 Πίνακας παραδοτέων

#### 4.4 Τεκμηρίωση λογισμικού

##### 4.4.1 Περιγραφή των απαιτήσεων από το λογισμικό

##### 4.4.2 Περιγραφή του σχεδίου του λογισμικού

##### 4.4.3 Εγχειρίδια χρήστη

##### 4.4.4 Πλάνο ελέγχου λογισμικού

##### 4.4.5 Αναφορές ελέγχου λογισμικού

#### 4.5 Τεκμηρίωση λοιπών παραδοτέων

#### 4.6 Ταξινόμηση παραδοτέων

### **5 Διαδικασίες ανάπτυξης λογισμικού**

#### 5.1 Σκοπός

#### 5.2 Μοντέλο κύκλου ζωής λογισμικού

#### 5.3 Βήματα ανάπτυξης λογισμικού

#### 5.4 Προδιαγραφή λογισμικού

#### 5.5 Σχεδίαση λογισμικού

#### 5.6 Χαρακτηριστικά του πηγαίου κώδικα

### **6 Έλεγχος υποπρομηθευτών**

#### 6.1 Διαδικασία ελέγχου και αποδοχής λογισμικού

#### 6.2 Διαδικασία διαχείρισης δραστηριοτήτων που σχετίζονται με υπηρεσίες προμηθευτών

### **7 Εσωτερικός έλεγχος και επισκοπήσεις παραδοτέων εγγράφων**

### **8 Εκπαίδευση – ενημέρωση ομάδας έργου**

## Δραστηριότητα I/Κεφάλαιο I3

---

Υπάρχουν τρεις βασικοί λόγοι που συνηγορούν υπέρ της χρησιμότητας τέτοιων προτύπων: πρώτον, ενσωματώνουν εμπειρία σχετικά με την καλύτερη πρακτική ανάπτυξης λογισμικού, δεύτερον, περιγράφουν τον τρόπο εφαρμογής των εργασιών διασφάλισης ποιότητας στην πράξη και, τρίτον, αποτελούν τη βάση για την ευέλικτη εκτέλεση των εργασιών από εναλλακτικές ομάδες ή άτομα σύμφωνα με τον τρόπο που αυτά περιγράφουν.

Είναι αναπόφευκτο, ασφαλώς, να σκεφτείτε ότι η δημιουργία και η εφαρμογή στην πράξη τέτοιων προτύπων προσθέτουν κόστος και καθυστερήσεις στην κατασκευή του λογισμικού. Δεν θα διαφωνήσουμε, θα σας κατευθύνουμε όμως στο να αναλογιστείτε αν η διασφάλιση καλύτερης ποιότητας του τελικού προϊόντος αποτελεί επαρκές αντιστάθμισμα για το κόστος αυτό.

## ΒΙΒΛΙΟΓΡΑΦΙΑ

*IEEE Guide for Software Quality Assurance Plans*, ANSI/IEEE, Std 730.I-1989

*IEEE Standard for Software Quality Assurance Plans*, ANSI/IEEE, Std 730-1989.

Pressman, R. S., *Software Engineering-A Practitioners Approach*, McGraw-Hill.

Sommerville, I. *Software Engineering*, London: Addison-Wesley.