Αποσπάσματα από το βιβλίο

Douglas Bell, Ian Morey, John Pugh, *The essence of program design*, Prentice Hall, 1997

ΚΕΦΑΛΑΙΟ 1 ΕΙΣΑΓΩΓΗ

1.1 Τι είναι η σχεδίαση (design);

Η σχεδίαση υπάρχει εδώ και χιλιάδες χρόνια σε πολλούς τομείς της ανθρώπινης δραστηριότητας. Σχεδιάζουμε

- αντικείμενα για οικιακή χρήση, όπως κρεβάτια, ηλεκτρικές κουζίνες και ηλεκτρικά πλυντήρια,
- συγκοινωνιακά συστήματα, όπως αυτοκίνητα και σιδηροδρομικά δίκτυα,
- κτίρια, όπως σπίτια και καθεδρικούς ναούς,
- μηχανές, όπως σταθμοί παραγωγής ηλεκτρικής ενέργειας και διυλιστήρια πετρελαίου.

Ιστορικά, οι διάφοροι τομείς σχεδίασης αναπτύχθηκαν ανεξάρτητα. Ο καθένας έχει τη δική του ομάδα από ειδικούς και δικές του τεχνικές και μεθόδους. Ο τρόπος με τον οποίο σχεδιάζεται ένα πλυντήριο είναι πολύ διαφορετικός από εκείνον με τον οποίο σχεδιάζεται μια γέφυρα πάνω από ένα ποτάμι.

Τι είναι κοινό για τη σχεδίαση σε όλους τους διαφορετικούς τομείς;

- Είναι δημιουργική – πριν δεν υπάρχει τίποτα, κατόπιν υπάρχει κάτι.
- Χρησιμοποιεί την επιστήμη όποτε κριθεί απαραίτητο.
- Οδηγεί σε κάτι που είναι χρήσιμο, ή ευχάριστο, ή και τα δύο.

Τι είναι η **σχεδίαση προγράμματος** (program design); Εσύ, ο αναγνώστης, γνωρίζεις τι είναι ένα πρόγραμμα υπολογιστή, και πιθανώς έχεις γράψει κάποια προγράμματα. Ξέρεις ότι το να κάνεις ένα πρόγραμμα να δουλέψει σωστά είναι μια πρόκληση και επηρεάζεται πολύ από τη δομή του προγράμματος. Για να ασχοληθείς με τη σχεδίαση προγράμματος πρέπει να προετοιμάσεις και να επινοήσεις τη δομή του προγράμματος. Η σχεδίαση προγράμματος αφορά ακριβώς τη δομή του προγράμματος. Η σχεδίαση προγράμματος αρχίζει με ένα προσδιορισμό του τι θα είναι ικανό να κάνει το πρόγραμμα και τελειώνει με τη δομή του προγράμματος.

Μελετούμε τη δομή ενός προγράμματος σε διάφορα επίπεδα λεπτομέρειας. Η συνολική δομή αποτελείται από έναν αριθμό συναρτήσεων (functions), διαδικασιών (procedures) ή υποπρογραμμάτων (subprograms). Το ακριβές όνομα αυτών των μονάδων διαφέρει από την μια προγραμματιστική γλώσσα στην άλλη. Υπάρχει μια πιο λεπτομερής δομή μέσα σε καθεμιά από αυτές τις μονάδες. Αυτή η δομή εκφράζεται με τις μεταβλητές και τις εντολές. Οι εντολές συνδυάζονται σε ακολουθιακή εκτέλεση, σε δομές επιλογής (εντολές `if`) και σε βρόχους επανάληψης (loops).

Μερικοί άνθρωποι αρνούνται την ύπαρξη της σχεδίασης. Λένε ότι «είναι φανερό» πώς πρέπει να μοιάζει ένα πρόγραμμα και επομένως δεν υπάρχει ανάγκη για οποιαδήποτε διαδικασία σχεδίασης. Εμείς διαφωνούμε. Η διαδικασία της σχεδίασης μπορεί να είναι αόρατη ή να γίνεται με το ένστικτο, αλλά υπάρχει!

Σε τι διαφέρει η σχεδίαση των προγραμμάτων υπολογιστή από τη σχεδίαση άλλων πραγμάτων;

- Υπάρχουν μέθοδοι για τη σχεδίαση προγράμματος.
- Το προϊόν της σχεδίασης προγράμματος δεν έχει απτή και υλική υπόσταση.

Διακρίνουμε δύο κλίμακες στη σχεδίαση προγράμματος. Η **αρχιτεκτονική ή υψηλού επιπέδου σχεδίαση** αναφέρεται στην δομή του προγράμματος ιδωμένη σε μεγάλη κλίμακα. Το προϊόν αυτής της φάσης είναι η δομή του λογισμικού, εκφρασμένη σε επίπεδο αρθρωμάτων (modules) και των διασυνδέσεών τους. Η **λεπτομερής ή χαμηλού επιπέδου σχεδίαση** σχετίζεται με τη λεπτομερή δομή για το κάθε άρθρωμα. Επίσης, η σχεδίαση χαμηλού επιπέδου σχετίζεται με τη λεπτομερή σχεδίαση προγραμμάτων μικρού έως μεσαίου μεγέθους.

1.2 Γιατί κάνουμε σχεδίαση προγράμματος;

Υπάρχουν αρκετοί λόγοι για να ισχυριστούμε ότι η σχεδίαση προγράμματος είναι σημαντική:

- Αξιοπιστία
- Κόστος
- Συντήρηση

Τα προγράμματα είναι σύνθετα. Αν συγκρίνεις την πολυπλοκότητα ενός προγράμματος και κάποιων άλλων αντικειμένων που σχεδιάζουμε –ενός αυτοκινήτου, ενός πλυντηρίου, ακόμα και ενός ηλεκτρονικού υπολογιστή– το πρόγραμμα είναι σαφέστατα πιο πολύπλοκο. Ως αποτέλεσμα, είναι δύσκολο να κατασκευαστεί πρόγραμμα που να δουλεύει χωρίς να σφάλει και να κάνει το αναμενόμενο. Υπάρχουν διάφοροι τρόποι για να βεβαιωθείς ότι ένα πρόγραμμα δουλεύει αξιόπιστα, αλλά το κλειδί είναι στη σχεδίαση. Αν ένα πρόγραμμα έχει σχεδιαστεί όπως πρέπει, θα πρέπει να λειτουργήσει και όπως πρέπει.

Ένα άλλο αποτέλεσμα της πολυπλοκότητας των προγραμμάτων είναι η συγγραφή τους παίρνει πολύ χρόνο και επομένως, εφόσον κάποιοι πληρώνονται, η ανάπτυξη προγραμμάτων κοστίζει πολλά χρήματα. Γενικά, το μεγαλύτερο διάστημα που αφιερώνεται στην ανάπτυξη ενός προγράμματος, καταναλώνεται στον έλεγχο, και ειδικότερα στην **αποσφαλμάτωση** (debugging). Ωστόσο, αν ένα πρόγραμμα σχεδιαστεί σωστά, θα είναι ευκολότερο να ελεγχθεί και να αποσφαλματωθεί.

Τα πιο χρήσιμα προγράμματα τροποποιούνται, αφού ολοκληρωθούν και κατά τη διάρκεια της χρήσης τους. Αυτό ονομάζεται **συντήρηση** (maintenance) προγράμματος. Επαγγελματίες προγραμματιστές ξοδεύουν πολύ χρόνο στην συντήρηση προγραμμάτων. Είναι μια απαιτητική και δύσκολη δουλειά γιατί οι συντηρητές-προγραμματιστές πρέπει να ανακαλύπτουν πώς δουλεύει το πρόγραμμα, ποια μέρη θέλουν αλλαγή και πώς να μεταβάλλουν το πρόγραμμα. Ένα πρόγραμμα που έχει σχεδιαστεί καλά είναι πιο εύκολο να τροποποιηθεί.

Εύλογα, η σχεδίαση προγράμματος είναι μια από τις πιο σημαντικές φάσεις στην ανάπτυξη του προγράμματος. Κατατάσσεται στο ίδιο επίπεδο σημασίας με τη φάση όπου προδιαγράφεται επακριβώς τι θέλουν οι χρήστες από το πρόγραμμα – αυτή η φάση καλείται ορισμός απαιτήσεων (requirements definition). Αν η σχεδίαση είναι καλή, όλες οι επόμενες φάσεις της ανάπτυξης θα πάνε καλά – η κωδικοποίηση, ο έλεγχος, η αποσφαλμάτωση και η προσαρμογή του προγράμματος στις μεταβαλλόμενες ανάγκες.

Αν προσεγγίσουμε τη σχεδίαση με συστηματικό τρόπο, θα δημιουργήσουμε ένα προϊόν αξιόπιστο, που θα ικανοποιεί τις προδιαγραφές του και θα είναι προσαρμόσιμο. Αν η σχεδίαση γίνει πρόχειρα, τότε όλες οι μεταγενέστερες φάσεις κατά πάσα πιθανότητα θα είναι δυσκολότερες και θα απαιτούν περισσότερο χρόνο. Το προϊόν που θα προκύψει δεν θα ικανοποιεί τις απαιτήσεις, θα είναι αναξιόπιστο και μη προσαρμόσιμο.

ΚΕΦΑΛΑΙΟ 9 ΚΑΤΕΥΘΥΝΤΗΡΙΕΣ ΓΡΑΜΜΕΣ ΓΙΑ ΤΗ ΣΧΕΔΙΑΣΗ ΠΡΟΓΡΑΜΜΑΤΟΣ

9.1 Εισαγωγή

Υπάρχουν αρκετές μέθοδοι για τη σχεδίαση προγράμματος. Μερικές μάλιστα διατείνονται ότι μπορούν να αποτελέσουν τη μοναδική και ιδεώδη προσέγγιση, αλλά στην πραγματικότητα καμιά μέθοδος δεν δικαιούται να ισχυρίζεται ότι οδηγεί τον σχεδιαστή σε μια ιδανική δομή προγράμματος. Με αυτό το δεδομένο, έχουν προκύψει κάποιες σημαντικές κατευθυντήριες γραμμές. Ο ρόλος τους είναι να συμπληρώνουν τις σχεδιαστικές μεθόδους παρέχοντας καθοδήγηση κατά τη διαδικασία της σχεδίασης. Επιπλέον, μας δίνουν τη δυνατότητα να κρίνουμε την ποιότητα του λογισμικού που έχει σχεδιαστεί. Αυτές οι κατευθυντήριες γραμμές εν πολλοίς είναι ανεξάρτητες από συγκεκριμένες σχεδιαστικές μεθόδους.

Αρκετές από τις κατευθυντήριες γραμμές επικεντρώνονται στην ιδέα της **αρθρωσιμότητας** (modularity). Η αρθρωσιμότητα είναι κύριος παράγοντας στη σχεδίαση προγράμματος. Έχει να κάνει με τη δομή του προγράμματος. Η δομή είναι το τελικό προϊόν όλων των μεθόδων σχεδίασης λογισμικού, όπως η λειτουργική αποσύνθεση (functional decomposition), η αντικειμενοστρεφής σχεδίαση (object-oriented design) και η σχεδίαση βάσει δομής δεδομένων (data structure design).

Αν μας ζητούσαν να σχεδιάσουμε ένα αυτοκίνητο, θα το σχεδιάζαμε πιθανόν τμηματικά από μερικά υποσυστήματα – μηχανή, σύστημα μετάδοσης, φρένα, αμάξωμα, κτλ. Ας ονομάσουμε αυτά τα υποσυστήματα αρθρώματα (modules). Κατά τον προσδιορισμό των συγκεκριμένων αρθρωμάτων ενός αυτοκινήτου διαλέξαμε τμήματα που είναι όσο το δυνατόν πιο ανεξάρτητα από τα υπόλοιπα. Αυτή είναι η ουσία της καλής αρθρωσιμότητας.

Οι κατευθυντήριες γραμμές που θα δώσουμε βοηθούν να απαντήσουμε ερωτήσεις, όπως:

- Πόσο μεγάλο μπορεί να είναι ένα άρθρωμα;
- Πόσο περίπλοκο είναι ένα άρθρωμα;
- Πώς μπορούμε να ελαχιστοποιήσουμε τις αλληλεπιδράσεις ανάμεσα στα αρθρώματα;

Πριν αντιμετωπίσουμε αυτά τα ερωτήματα πρέπει να προσδιορίσουμε τις έννοιες "άρθρωμα" και "αρθρωσιμότητα". Συνήθως αυτό υπαγορεύεται από πρακτικούς παράγοντες, όπως οι δυνατότητες που παρέχει η διαθέσιμη γλώσσα προγραμματισμού και το λειτουργικό σύστημα. Υπάρχει ποικιλία από μηχανισμούς για την κατάτμηση του λογισμικού σε ανεξάρτητα αρθρώματα, ή, με άλλα λόγια, για την ομαδοποίηση στοιχείων που παρουσιάζουν συνάφεια. Σε ποικίλες γλώσσες προγραμματισμού, το άρθρωμα καλείται:

COBOL	υποπρόγραμμα (subprogram)
BASIC	υπορουτίνα (subroutine)
FORTRAN	υπορουτίνα
Pascal	διαδικασία (procedure) ή συνάρτηση (function)
Modula 2	άρθρωμα (module)
C	συνάρτηση
C++	αντικείμενο (object)
Ada	διαδικασία ή συσκευασία (package)

Έτσι ένα άρθρωμα είναι ένα σχετικά ανεξάρτητο τμήμα προγράμματος που συνήθως έχει ένα όνομα και μερικές εντολές. Ένα άρθρωμα καλείται από κάποιο άλλο άρθρωμα και ενδεχομένως χρησιμοποιεί (καλεί) άλλα αρθρώματα. Σε αυτό το κεφάλαιο χρησιμοποιούμε τον όρο άρθρωμα με τον πιο γενικό τρόπο για να περιλαμβάνει τρέχοντες ή μελλοντικούς μηχανισμούς που οργανώνουν το λογισμικό σε εύχρηστα τμήματα.

9.2 Γιατί αρθρωσιμότητα;

Κατά κανόνα, το λογισμικό αποτελείται από χιλιάδες γραμμές κώδικα. Τέτοια συστήματα παρουσιάζουν ασύλληπτη πολυπλοκότητα. Απαιτούνται τεχνικές αντιμετώπισης της πολυπλοκότητας. Ουσιαστικά, ο στόχος για την αρθρωσιμότητα είναι η κατασκευή λογισμικού από τμήματα που θα είναι όσο το δυνατόν πιο ανεξάρτητα μεταξύ τους. Ιδανικά, κάθε άρθρωμα πρέπει να είναι αυτόνομο και να περιέχει όσο το δυνατόν λιγότερες αναφορές σε άλλα αρθρώματα. Αυτός ο στόχος έχει συνέπειες σε όλα τα στάδια της ανάπτυξης λογισμικού, όπως αναλύεται στις επόμενες παραγράφους.

Αρχιτεκτονική σχεδίαση
Αυτό είναι το βήμα κατά τη διάρκεια του οποίου καθορίζεται μεγάλο μέρος της δομής του λογισμικού. Είναι επομένως κρίσιμο για τη δημιουργία καλής αρθρωσιμότητας. Μια σχεδιαστική προσέγγιση που οδηγεί σε φτωχή αρθρωσιμότητα θα οδηγήσει τρομερές συνέπειες αργότερα.

Σχεδίαση αρθρωμάτων
Αν η αρχιτεκτονική σχεδίαση είναι αρθρωτή, τότε η σχεδίαση αρθρωμάτων θα είναι εύκολη. Κάθε άρθρωμα θα έχει ένα μοναδικό και καλώς προσδιορισμένο σκοπό με λίγες, καθαρές διασυνδέσεις με άλλα αρθρώματα.

Αποσφαλμάτωση
Η σημασία της αρθρωσιμότητας αποδεικνύεται κατά τη διάρκεια της αποσφαλμάτωσης. Αν η δομή είναι αρθρωτή, θα είναι εύκολο να προσδιοριστεί ποιο συγκεκριμένο άρθρωμα είναι υπεύθυνο για το παρατηρούμενο σφάλμα. Όμοια, η διόρθωση σε ένα άρθρωμα δεν θα έχει συνέπειες σε άλλα αρθρώματα.

Έλεγχος
Ο έλεγχος ενός μεγάλου συστήματος φτιαγμένου από ένα μεγάλο αριθμό αυτοτελών αρθρωμάτων είναι δύσκολος και χρονοβόρος. Είναι πρακτικά αδύνατον να ελεγχθεί το πλήρες σύστημα ως ενιαίο και συνήθως οι δοκιμές γίνονται σε κάθε τμήμα ξεχωριστά. Γι' αυτό είναι κρίσιμη η δομή του συστήματος.

Συντήρηση
Συντήρηση σημαίνει απαλοιφή των σφαλμάτων (bugs) και βελτίωση του συστήματος, ώστε να ανταποκρίνεται στις μετα-

βαλλόμενες απαιτήσεις του χρήστη. Αυτή η δραστηριότητα καταναλώνει τεράστιες ποσότητες του χρόνου ανάπτυξης. Πάλι, η αρθρωσιμότητα είναι σημαντική. Το ιδανικό είναι, κάνοντας μια αλλαγή σε ένα άρθρωμα, να είσαι βέβαιος ότι κανένα άλλο δεν πρόκειται να επηρεαστεί. Πολύ συχνά, οι φανερές ή διακριτικές διασυνδέσεις ανάμεσα σε αρθρώματα κάνουν την συντήρηση έναν εφιάλτη.

Υπάρχουν δύο ακόμη σημεία που επιτείνουν τη σημασία της αρθρωσιμότητας.

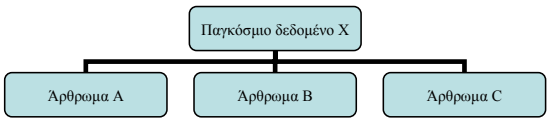
Ανεξάρτητη ανάπτυξη
Τα περισσότερα προγράμματα υλοποιούνται από ομάδες ανθρώπων που εργάζονται επί μήνες ή χρόνια. Φυσιολογικά, κάθε άρθρωμα αναπτύσσεται από ένα μόνο άτομο. Είναι ζωτικό οι διεπαφές (interfaces) ανάμεσα στα αρθρώματα να είναι καθαρές και μικρές.

Επαναχρησιμοποίηση λογισμικού
Μια κύρια κατεύθυνση είναι η **επαναχρησιμοποίηση** λογισμικού από προηγούμενα προγραμματιστικά έργα. Έτσι αποφεύγεται η επανεφεύρεση του τροχού και ενδεχομένως γλιτώνουμε από κατασπατάληση πόρων. Ωστόσο, ένα άρθρωμα δεν μπορεί να επαναχρησιμοποιηθεί εύκολα, αν συνδέεται με πολύπλοκο τρόπο με άλλα αρθρώματα στο προηγούμενο σύστημα. Μια μεταμόσχευση καρδιάς θα ήταν αδύνατη αν υπήρχαν πολλές αρτηρίες, φλέβες, νεύρα κτλ να κοπούν και να επανασυνδεθούν.

9.5 Τα παγκόσμια δεδομένα είναι επιβλαβή

Όπως η διαβόητη εντολή `goto` έπεσε σε ανυποληψία, έτσι και οι τρέχουσες απόψεις της τεχνολογίας λογισμικού (software engineering) θεωρούν ότι τα παγκόσμια (ή καθολικά) δεδομένα είναι επιβλαβή. Πριν συζητήσουμε τα επιχειρήματα, ας σώσουμε μερικούς ορισμούς. Με τον όρο **παγκόσμια δεδομένα** (global data), εννοούμε δεδομένα που χρησιμοποιούνται ευρέως σε όλη την έκταση ενός προγράμματος και γι' αυτό είναι προσπελάσιμα από όλα τα αρθρώματα του συστήματος. Συνήθως, τα παγκόσμια δεδομένα δηλώνονται στην αρχή του προγράμματος. Με τον όρο **τοπικά δεδομένα** (local data), εννοούμε δεδομένα που μπορούν να χρησιμοποιηθούν μόνον μέσα σε συγκεκριμένα υποπρογράμματα – η πρόσβαση σε αυτά είναι ελεγχόμενη. Τα τοπικά δεδομένα συνήθως δηλώνονται μέσα στο υποπρόγραμμα που τα χρησιμοποιεί.

Για κάθε κομμάτι λογισμικού, ο σχεδιαστής μπορεί να επιλέξει εάν θα κάνει τα δεδομένα παγκόσμια ή τοπικά. Βέβαια, δεδομένα μπορούν επίσης να περνιούνται ανάμεσα στα υποπρογράμματα μέσω του μηχανισμού περάσματος παραμέτρων (parameter passing).



Ας δούμε τι «τρέχει» με τα παγκόσμια δεδομένα. Ας υποθέσουμε ότι τρία αρθρώματα ονομαζόμενα A, B, C προσπελαύνουν κάποια παγκόσμια δεδομένα, όπως φαίνεται στο σχήμα. Ας υποθέσουμε ότι πρέπει να μελετήσουμε το άρθρωμα A για να κάνουμε μια αλλαγή σε αυτό. Έστω ότι και τα δύο αρθρώματα A, B προσπελαύνουν το παγκόσμιο δεδομένο που ονομάζεται X. Τότε, για να καταλάβουμε το A, πρέπει να καταλάβουμε το ρόλο του X. Αλλά, τώρα, για να καταλάβουμε το X πρέπει να εξετάσουμε το B. Έτσι καταλήγουμε να πρέπει να μελετήσουμε ένα δεύτερο άρθρωμα (B), ενώ το μόνο που θέλαμε ήταν να καταλάβουμε ένα. Αλλά η ιστορία χειροτερεύει. Υποθέτουμε ότι τα αρθρώματα B και C μοιράζονται κάποια δεδομένα. Έτσι, για να καταλάβουμε πλήρως B θα πρέπει να καταλάβουμε το C, κ.ο.κ. Βλέπουμε ότι για να κατανοήσουμε κάθε άρθρωμα που χρησιμοποιεί παγκόσμια δεδομένα πρέπει να καταλάβουμε όλα τα αρθρώματα που τα χρησιμοποιούν.

Γενικά, τα τοπικά δεδομένα είναι προτιμότερα διότι γιατί κάνουν ευκολότερη:

- την ανάγνωση και τη μελέτη ενός υποπρογράμματος, επειδή είναι φανερό τι δεδομένα χρησιμοποιεί.
- την επαναχρησιμοποίηση ενός υποπρογράμματος σε ένα νέο πρόγραμμα, επειδή είναι ένα αυτόνομο πακέτο.
- την ανάγνωση και την κατανόηση των παγκόσμιων δεδομένων (αν υπάρχουν), επειδή έχουν περιοριστεί σε πλήθος.

Επομένως, γενικά, τα παγκόσμια δεδομένα πρέπει να ελαχιστοποιηθούν (ή να καταργηθούν) και τα τοπικά δεδομένα να αυξηθούν. Ωστόσο, τέτοιοι απόλυτοι κανόνες δεν πρέπει να ακολουθούνται τυφλά – υπάρχουν πάντα περιστάσεις που επιτρέπεται να τους παραβιάζουμε. Για παράδειγμα, σε κάποιες περιπτώσεις μπορεί να είναι προτιμότερο να χρησιμοποιηθεί μια παγκόσμια μεταβλητή, παρά να περαστούν πελώριες δομές ως παράμετροι.

9.6 Απόκρυψη πληροφοριών και αφαιρετικοί τύποι δεδομένων

Η **απόκρυψη πληροφοριών** (information hiding), ή απόκρυψη δεδομένων (data hiding), ή **ενθυλάκωση** (encapsulation) όπως ονομάζεται μερικές φορές, είναι μια μέθοδος για τη δόμηση του λογισμικού. Η κεντρική ιδέα είναι ότι για κάθε δομή δεδομένων (ή δομή αρχείων),

- η ίδια η δομή,
- οι εντολές που προσπελαύνουν τη δομή, και
- οι εντολές που την τροποποιούν

πρέπει να απαρτίζουν ένα μοναδικό άρθρωμα. Ενθυλακωμένα δεδομένα δεν μπορεί να προσπελαύνονται κατευθείαν. Μπορεί να προσπελαύνονται μόνον μέσω των διαδικασιών (procedures) που συσχετίζονται με τα δεδομένα. Μία τέτοια συλλογή από δεδομένα και διαδικασίες ονομάζεται **αφαιρετικός τύπος δεδομένων** (abstract data type).

Ένα κλασικό παράδειγμα χρήσης της απόκρυψης πληροφοριών είναι η στοίβα (stack). Υπάρχουν διαδικασίες για την αρχικοποίηση της στοίβας, για την τοποθέτηση ενός στοιχείου στην κορυφή του σωρού και την αφαίρεση ενός στοιχείου από την κορυφή. (Προαιρετικά, προσφέρεται και μια συνάρτηση που ελέγχει εάν η στοίβα είναι άδεια.) Πρόσβαση στη στοίβα είναι δυνατή μόνον μέσω αυτών των διαδικασιών. Με δεδομένη αυτή την προδιαγραφή, ο προγραμματιστής μπορεί να υλοποιήσει τη στοίβα αποθηκεύοντάς την σε έναν πίνακα (array), σε μια συνδεδεμένη λίστα (linked list), ή οτιδήποτε άλλο. Ο χρήστης της στοίβας δεν χρειάζεται να ξέρει, ούτε και να νοιάζεται για το πώς υλοποιήθηκε η στοίβα. Οποιαδήποτε αλλαγή στην αναπαράσταση της στοίβας δεν έχει καμία επίπτωση στους χρήστες (εκτός, ίσως, από την απόδοση).

Η απόκρυψη πληροφοριών ικανοποιεί τρεις στόχους:

1. Ευκολία τροποποιήσεων: αν αλλάξει μια σχεδιαστική απόφαση, οι αλλαγές περιορίζονται σε λίγα αρθρώματα ή, εάν είναι δυνατόν, μόνον σε ένα.
2. Ανεξάρτητη ανάπτυξη: όταν ένα σύστημα αναπτύσσεται από μια ομάδα προγραμματιστών, οι διεπαφές μεταξύ των αρθρωμάτων πρέπει να είναι, όσο το δυνατόν, πιο απλές. Απόκρυψη πληροφοριών σημαίνει ότι οι διεπαφές είναι κλήσεις σε υποπρογράμματα. Αυτό είναι πιο απλό από την προσπέλαση σε διαμοιραζόμενα δεδομένα ή δομές αρχείων.
3. Κατανοησιμότητα: για λόγους σχεδίασης, ελέγχου, δοκιμής και συντήρησης, είναι σημαντικό να κατανοούμε κάθε άρθρωμα ανεξάρτητα από τα υπόλοιπα. Όπως έχουμε δει, τα παγκόσμια και τα διαμοιραζόμενα δεδομένα δε μας βοηθούν να καταλάβουμε το λογισμικό.

Ως παράδειγμα, ας υποθέσουμε ότι πρέπει να γράψουμε ένα πρόγραμμα μορφοποίησης κειμένου. Φτιάχνουμε κάθε γραμμή κειμένου τοποθετώντας χαρακτήρες, τον ένα μετά τον άλλο μέχρι να γεμίσει η γραμμή. Μπορούμε να επιτρέψουμε στο πρόγραμμα να προσεγγίσει την προσωρινή μνήμη (buffer) κατευθείαν, για παράδειγμα όταν χρειάζεται να τοποθετηθούν χαρακτήρες στην προσωρινή μνήμη. Θα μπορούσαμε κάλλιστα να τελειώσουμε με ένα πρόγραμμα στο οποίο από διάφορα σημεία του κώδικα θα προσπελαύνεται κατευθείαν η προσωρινή μνήμη. Εισάγοντας ένα χαρακτήρα στην προσωρινή μνήμη πρέπει πάντοτε να θυμόμαστε να επαυξάνουμε τον δείκτη ή τον μετρητή που ορίζει την επόμενη ελεύθερη θέση. Υπάρχουν μεγάλα περιθώρια λάθους, για παράδειγμα, μπορεί να ξεχάσουμε να ενημερώσουμε τον δείκτη ή να τον ενημερώσουμε πριν να τοποθετήσουμε τον χαρακτήρα.

Μία εναλλακτική δομή θα ήταν να δημιουργήσουμε τρεις διαδικασίες:

1. Αρχικοποίηση της προσωρινής μνήμης
2. Τοποθέτηση ενός χαρακτήρα στη μνήμη
3. Τερματισμός (εκτύπωση υπολειπόμενων γραμμών)

Τώρα καμιά από τις πληροφορίες σχετικά με τη γραμμή – το μήκος της, το όνομά της, ο δείκτης της ελεύθερης θέσης – δεν χρειάζεται να αναφερθεί πουθενά μέσα στο πρόγραμμα. Οποιαδήποτε αλλαγή στην αναπαράσταση της προσωρινής μνήμης, ή στο μήκος της γραμμής, είναι εντελώς περιορισμένη στο εσωτερικό των τριών διαδικασιών.

Ορισμένες γλώσσες προγραμματισμού (Ada, C++, Modula-2) δεν επιτρέπουν αναφορές προς το άρθρωμα, παρά μόνον κλήσεις προς τις τρεις διαδικασίες. Αυτό βοηθάει σε μεγάλο βαθμό στην αποφυγή σφαλμάτων. Σαφέστατα, τέτοια χαρακτηριστικά των γλωσσών προγραμματισμού βοηθούν στην κατασκευή λογισμικού σύμφωνα με την απόκρυψη πληροφοριών. Μια άλλη διευκόλυνση θα ήταν το άρθρωμα να αρχικοποιεί τον εαυτό του, οπότε δεν θα χρειαζόταν ο προγραμματιστής να καλέσει την πρώτη διαδικασία.

Σαν ένα πρόσθετο παράδειγμα, ας θεωρήσουμε ένα πληροφοριακό σύστημα, για παράδειγμα, ένα σύστημα κράτησης θέσεων σε αεροπορικές εταιρείες, που ανταποκρίνεται σε εισόδους από ένα τερματικό. Οι εντολές μπορεί να είναι οι εξής:

- Κράτηση θέσης
- Ακύρωση της κράτησης
- Ερώτηση για τη διαθεσιμότητα των θέσεων

Μια άποψη της δομής του προγράμματος θα ήταν να το βλέπαμε σαν δύο αρθρώματα:

1. ένα που αλληλεπιδρά με τον χρήστη μέσω του τερματικού, και
2. ένα που αποθηκεύει και ανακαλεί τις πληροφορίες από τη βάση δεδομένων.

Καθένα από τα παραπάνω μπορούμε να το δούμε σαν ένα άρθρωμα που περιβάλλει τα δεδομένα, παρέχοντας πρόσβαση σε αυτά μέσω κλήσεων διαδικασιών. Μπορούμε βέβαια να επινοήσουμε και ένα τρίτο άρθρωμα, σε ρόλο ελεγκτή που καλεί τα άλλα δύο, το ένα μετά το άλλο.

Συνοπτικά, η αρχή της απόκρυψης πληροφοριών σημαίνει ότι στο τέλος της σχεδιαστικής διαδικασίας, κάθε δομή δεδομένων ή αρχείο προσπελαύνεται μόνο μέσω συγκεκριμένων και σαφώς προσδιορισμένων διαδικασιών ή υποπρογραμμάτων. Δυστυχώς μόνον μερικές γλώσσες προγραμματισμού υποστηρίζουν απόκρυψη πληροφοριών.

Ασήμαντη αρχικά, η απόκρυψη πληροφοριών, εξελίχθηκε ραγδαία σε μια σημαντική έννοια στη σχεδίαση προγραμμάτων και στην τεχνολογία λογισμικού. Δεν επηρέασε μόνο τη σχεδίαση των γλωσσών προγραμματισμού, αλλά οδήγησε σε διαφορετικά πρότυπα προγραμματισμού, όπως τον αντικειμενοστρεφή προγραμματισμό.

ΚΕΦΑΛΑΙΟ 10 ΑΝΑΣΚΟΠΗΣΗ

10.1 Πάνω-κάτω ή κάτω-πάνω;

Σήμερα κυριαρχεί η μέθοδος σχεδιασμού πάνω-προς-κάτω (top-down), δηλαδή από τη γενική ιδέα στις λεπτομέρειες. Αρχίζει προσδιορίζοντας τα κύρια χαρακτηριστικά του προγράμματος. Σε ένα αρχικό επίπεδο, η σχεδίαση του προγράμματος χρησιμοποιεί κάποια καθορισμένα, αλλά προς το παρόν μη υλοποιημένα στοιχεία. Αυτά με τη σειρά τους χρησιμοποιούν άλλα στοιχεία χαμηλότερου επιπέδου, κοκ. Το πρόγραμμα μεγαλώνει όπως ένα (ανεστραμμένο) δέντρο από τις ρίζες του ως τα κλαδιά και τα φύλλα. Τα κύρια συστατικά του προγράμματος πρέπει να λαμβάνονται υπ' όψη από την αρχή. Οι λεπτομέρειες μπορούν να μελετηθούν αργότερα.

Η σχεδίαση κάτω-προς-πάνω (bottom-up), δηλαδή από τις λεπτομέρειες προς τη γενική ιδέα, είναι ακριβώς αντίθετη. Αρχίζει με τα στοιχεία του χαμηλότερου επιπέδου. Ο προγραμματιστής σκέφτεται ποια στοιχεία χαμηλού επιπέδου θα φανούν χρήσιμα στο τελικό πρόγραμμα, και τα κατασκευάζει το ένα μετά το άλλο. Έπειτα, τα στοιχεία αυτού του χαμηλού επιπέδου χρησιμοποιούνται για την κατασκευή του επιπέδου που βρίσκεται πιο πάνω, κοκ.

Στις μέρες μας, η μέθοδος σχεδίασης από τις λεπτομέρειες προς τη γενική ιδέα (bottom-up) έχει γίνει αντικείμενο κριτικής, γιατί αρχίζει από τις λεπτομέρειες, αντί από τα κύρια τμήματα του προγράμματος. Είναι σαν να αρχίζουμε να σχεδιάζουμε μια γέφυρα σκεπτόμενοι πρώτα πόσο μεγάλες πρέπει να είναι οι βίδες, ή να προετοιμάζουμε ένα γεύμα αποφασίζοντας τι αλάτι θα χρησιμοποιήσουμε, ιωδιούχο ή κανονικό. Αλλά αναμφισβήτητα και αυτή η μέθοδος σχεδίασης είναι χρήσιμη σε κάποιες περιπτώσεις. Ας υποθέσουμε ότι θέλουμε να δημιουργήσουμε ένα πρόγραμμα που παίζει σκάκι. Μπορούμε κάλ-

λιστα να φανταστούμε μερικές συναρτήσεις που θα είναι χρήσιμες. Για παράδειγμα,

- εμφάνισε τη σκακιέρα,
- έλεγξε αν υπάρχει αντικείμενο πάνω σε κάποιο τετράγωνο,
- έλεγξε αν μια συγκεκριμένη κίνηση είναι επιτρεπτή.

Αυτές οι συναρτήσεις θα ανήκουν στη λεπτομερειακή σχεδίαση, θα κατασκευαστούν δηλαδή με τη μέθοδο από τις λεπτομέρειες προς τη γενική ιδέα (bottom-up).

Αρκετοί υποστηρίζουν ότι συχνά είναι δύσκολο να ξεκινήσεις είτε από την γενική ιδέα, είτε από τις λεπτομέρειες του προγράμματος. Αντί αυτών, ένα μέσο σημείο του προγράμματος φαίνεται σαν μια προφανές σημείο εκκίνησης για να ξεκινήσει από εκεί η σχεδίαση. Το δέντρο ξεκινάει από τη μέση. Μεγαλώνει και από τις δυο πλευρές, ταυτόχρονα προς τη ρίζα και προς τα φύλλα. Αυτό ονομάζεται μέσα-προς-έξω (middle-out) σχεδίαση. Δεν χαίρει ιδιαίτερης εκτίμησης, αλλά αναμφισβήτητα χρησιμοποιείται στην πράξη.

Η κλίμακα για τις μεθόδους της σχεδίασης που έχει αποτυπωθεί μέχρι στιγμής φαίνεται ως εξής παρακάτω:

κάτω-προς-πάνω	μέσα-προς-έξω	πάνω-προς-κάτω
bottom-up	middle-out	top-down

10.2 Δεδομένα ή πράξεις;

Υπάρχει και μια άλλη διάσταση στη σχεδίαση - κατά πόσο πρέπει να βασίζεται στα δεδομένα ή στις πράξεις. Η **λειτουργική αποδόμηση** (functional decomposition) είναι μια μέθοδος πάνω-προς-κάτω (top-down), δηλαδή από τη γενική ιδέα στη λεπτομέρεια, που επικεντρώνεται στις λειτουργίες (ή πράξεις) που πρέπει να εκτελέσει το πρόγραμμα. Η σχεδίαση ξεκινά με τις κυριότερες συνολικά λειτουργίες που πρέπει να κάνει το πρόγραμμα. Οι πιο ειδικές λειτουργίες υλοποιούνται ακολουθώντας το πρότυπο γενικό προς ειδικό (top-down). Τα δεδομένα αντιμετωπίζονται απλώς ως παραπροϊόν, όταν και όπου είναι απαραίτητο.

Στην απέναντι πλευρά του φάσματος βρίσκεται η σχεδίαση που βασίζεται στη δομή των δεδομένων που επεξεργάζεται το πρόγραμμα. Αυτό ονομάζεται **σχεδίαση βάσει δομής δεδομένων** (data structure design), ή μέθοδος Jackson Structured Programming. Αρχίζει αναλύοντας τη δομή των δεδομένων εισόδου και εξόδου πάνω στα οποία πρόκειται να λειτουργήσει το πρόγραμμα. Η δομή του προγράμματος προκύπτει από τη δομή των δεδομένων σε μια σειρά από καθορισμένα βήματα.

Ανάμεσα στους δυο πόλους της λειτουργικής αποδόμησης και της σχεδίασης βάσει δομής δεδομένων βρίσκεται η **αντικειμενοστρεφής σχεδίαση** (object-oriented design). Σε αυτή τη μέθοδο αναγνωρίζεται η συμβιωτική σχέση μεταξύ δεδομένων και πράξεων. Τα δεδομένα μαζί με τις λειτουργίες που επιδρούν πάνω σε αυτά ενθυλακώνονται σε ενότητες που ονομάζονται αντικείμενα (objects). Η σχεδίαση προχωράει με την αναγνώριση των αντικειμένων στο πρόγραμμα και των σχέσεων μεταξύ τους. Επειδή δεδομένα και πράξεις έχουν εξίσου σημαντικό ρόλο, μπορούμε να υποστηρίξουμε ότι η αντικειμενοστρεφής σχεδίαση αποτελεί μια ολιστική προσέγγιση στη σχεδίαση.

Το φάσμα των σχεδιαστικών προσεγγίσεων, σύμφωνα με τη σχέση των δεδομένων και πράξεων είναι:

Λειτουργική αποδόμηση	Αντικειμενοστρεφής σχεδίαση	Σχεδίαση βάσει δομής δεδομένων
functional decomposition	object-oriented design	data structure design

10.3 Τα τρία συστατικά των προγραμμάτων

Η ανάπτυξη προγράμματος ξεκινά με την καταγραφή των απαιτήσεων του χρήστη και προσπαθεί να τις ικανοποιήσει μέσω της δημιουργίας ενός προγράμματος. Ευλόγως, το αντικείμενο κάθε μεθόδου σχεδίασης προγράμματος είναι να συλλάβει και να μοντελοποιήσει τις πιο σημαντικές πλευρές των απαιτήσεων. Οι απαιτήσεις του χρήστη αποτυπώνονται συνήθως (αλλά όχι πάντα!) με τη μορφή ενός συνόλου προδιαγραφών. Κατά την ανάπτυξη προγράμματος οι απαιτήσεις με κάποιο τρόπο μετασχηματίζονται σε μια προγραμματιστική δομή. Ο συγκεκριμένος τρόπος με τον οποίο γίνεται αυτός ο μετασχηματισμός διαφοροποιεί την μια μέθοδο σχεδίασης από την άλλη.

Ένα τυπικό, διαδικαστικό (procedural) πρόγραμμα αποτελείται από τρία συστατικά:

- Ορισμένα δεδομένα (αριθμοί, χαρακτήρες, πίνακες, πιο περίπλοκες δομές)
- Ορισμένες πράξεις πάνω στα δεδομένα (υπολογισμοί, είσοδος, έξοδος, συγκρίσεις)
- Συγκεκριμένο έλεγχο του συγχρονισμού των πράξεων (ακολουθία, επαναλήψεις, επιλογές)

Ο συγχρονισμός των πράξεων ρυθμίζει τη σειρά με την οποία διεξάγονται οι πράξεις, εάν επαναλαμβάνονται ή όχι, και εάν εκτελούνται υπό συνθήκες ή πάντοτε.

Κάποια από τα τρία αυτά συστατικά συνήθως εμφανίζονται κατά τη διατύπωση του προβλήματος, δηλαδή στις προδιαγραφές. Ωστόσο, συχνά μερικά συστατικά δεν παρουσιάζονται στο πρόβλημα, και πρέπει να ενσωματωθούν από τον προγραμματιστή κατά την υλοποίηση της λύσης που έχει επινοήσει.

Μερικές μέθοδοι σχεδίασης προγραμμάτων διατείνονται ότι απευθύνονται και στα τρία προαναφερόμενα συστατικά. Στην πραγματικότητα όμως, αρκετές μέθοδοι επικεντρώνονται μόνο σε ένα συστατικό, υποβαθμίζοντας τα άλλα δύο. Μια τέτοια μέθοδος τείνει να είναι επιτυχημένη σε εφαρμογές όπου η σπουδαιότητα του συγκεκριμένου συστατικού είναι υψηλή.

Θα κάνουμε τώρα μια ανασκόπηση σε περιοχές εφαρμογών της πληροφορικής και θα αναγνωρίσουμε για κάθε περιοχή, ποιο συστατικό είναι σημαντικό:

- Σε εφαρμογές πραγματικού χρόνου (real-time) ή εφαρμογές ελέγχου διεργασιών (process control), υπάρχει συχνά μια περίπλοκη σχέση ανάμεσα στα συμβάντα (events). Το καίριο στοιχείο είναι πότε διεξάγονται τα συμβάντα. Τα δεδομένα είναι απλά ή ουσιαστικά ανύπαρκτα.
- Σε ένα επιστημονικό ή μαθηματικό πρόγραμμα, τα δεδομένα είναι συχνά απλά, όμως η αλγοριθμική πολυπλοκότητα είναι τεράστια.
- Σε επιχειρησιακές εφαρμογές, που παλιότερα λέγονταν μηχανογραφικές εφαρμογές, τα δεδομένα είναι πλούσια στη δομή, αλλά οι απαιτήσεις για επεξεργασία τους είναι μικρές.

Ας δούμε τώρα μερικές μεθόδους σχεδίασης που περιγράφηκαν:

- Η λειτουργική αποδόμηση επικεντρώνεται στις πράξεις και στην ακολουθία των πράξεων. Τα δεδομένα εν πολλοίς αγνοούνται.
- Η σχεδίαση βάσει δομής δεδομένων επικεντρώνεται στα δεδομένα. Θεωρείται ότι η ακολουθία των πράξεων αντικατοπτρίζει τη δομή των δεδομένων. Έτσι η ακολουθία προκύπτει από τη δομή δεδομένων.
- Στον αντικειμενοστρεφή σχεδιασμό, έχουμε σκοπό να απεικονίσουμε όσο το δυνατόν πιο πιστά τον πραγματικό κόσμο που μοντελοποιείται. Αυτό περιλαμβάνει και τα δεδομένα και τις πράξεις που συμμετέχουν.

Συνεπώς, διαφορετικές προσεγγίσεις στη σχεδίαση, δίνουν έμφαση σε διαφορετικές πλευρές του προβλήματος που πρόκειται να λυθεί.

10.4 Ο προγραμματισμός ως μοντελοποίηση

Μερικές μέθοδοι σχεδίασης προγραμμάτων επιχειρούν να «μοντελοποιήσουν» κάποιες πλευρές της κατάστασης που θέλουμε να προγραμματίσουμε. Θα δούμε τρεις προσεγγίσεις μοντελοποίησης που καταλήγουν στη δομή του προγράμματος. Οι τρεις αυτές προσεγγίσεις έχουν ως κοινό στοιχείο ότι διατείνονται ότι μοντελοποιούν με πιστότητα την κατάσταση που θέλουμε να προγραμματίσουμε.

Αρχικά, ας θεωρήσουμε ένα πρόγραμμα που ελέγχει ένα πλυντήριο. Γραμμένο σε μορφή ψευδοκώδικα, ένα μέρος του, θα έχει την εξής παρακάτω μορφή:

- Άνοιξε τη βαλβίδα του νερού
- Περίμενε μέχρι να γεμίσει το δοχείο
- Περίμενε 5 λεπτά

Αυτό το πρόγραμμα περιγράφει λειτουργίες του πλυντηρίου. Η καταλληλότερη μέθοδος σχεδίασης θα ήταν η λειτουργική αποδόμηση.

Τώρα, ας σκεφτούμε ένα πρόγραμμα το οποίο προσθέτει μια λίστα με αριθμούς (ας πούμε τιμές μια λίστα με ψώνια) που πληκτρολογείται στον υπολογιστή. Οι αριθμοί επαναλαμβάνονται, οπότε θα χρειαστούμε και μια αντίστοιχη επανάληψη στο πρόγραμμα. Το πρόγραμμα θα είναι κάπως έτσι:

```
θέσε το άθροισμα στο μηδέν
επαναλάμβανε
        διάβασε έναν αριθμό
        πρόσθεσέ τον στο άθροισμα
μέχρις ότου να μην υπάρχουν άλλοι αριθμοί
πρόβαλε το άθροισμα
```

Εδώ το πρόγραμμα ακολουθεί τη δομή των δεδομένων εισόδου. Η καταλληλότερη μέθοδος σχεδίασης είναι αυτή της σχεδίασης βάσει δομής δεδομένων.

Μελετώντας τα δυο προηγούμενα παραδείγματα και τη μέθοδο που ακολουθήσαμε για την κατασκευή τους, παρατηρούμε ότι ενώ και τα δυο επιχειρούν να μοντελοποιήσουν πραγματικές, απτές καταστάσεις, το ένα χρησιμοποιεί τις λειτουργίες ως σημείο εκκίνησης, ενώ το άλλο χρησιμοποιεί τα δεδομένα. Το πρώτο πρόγραμμα μοιάζει με προγράμματα μαθηματικών ή ελέγχου διεργασιών, όπου η δομή της εισόδου ή της εξόδου της πληροφορίας είναι αμελητέα. Το δεύτερο πρόγραμμα, μοιάζει με μηχανογραφικές εφαρμογές, όπου τα δεδομένα έχουν πλούσια δομή. Πράγματι, σε ένα σύστημα με βάση δεδομένων, τα δεδομένα είναι στο επίκεντρο, ενώ τα προγράμματα που τα προσπελαύνουν παίζουν ένα δευτερεύοντα ρόλο.

Συνήθως, τα προγράμματα δεν ανήκουν σε καμιά από τις παραπάνω κατηγορίες. Ας πάρουμε για παράδειγμα, ένα πρόγραμμα που έχει να κάνει με διαδραστική είσοδο από τον υπολογιστή. Ο χρήστης έχει στη διάθεσή του ένα μενού με διαθέσιμες επιλογές που μπορεί να επιλέξει χρησιμοποιώντας το ποντίκι ή το πληκτρολόγιο. Εδώ τα δεδομένα δεν έχουν κάποια εμφανή δομή και δεν γνωρίζουμε ποια πράξη πρόκειται να εκτελεστεί. Έτσι θα κατασκευάζαμε ένα πρόγραμμα όπως το παρακάτω:

```
επαναλάμβανε
    όταν ο χρήστης κάνει μια επιλογή
    ανάλογα με την επιλογή του χρήστη
            εάν είναι αποθήκευση:
                    κάνε αποθήκευση
            εάν είναι εκτύπωση:
                    κάνε εκτύπωση
            εάν είναι διαγραφή:
                    κάνε διαγραφή της επιλογής
            εάν είναι τερματισμός:
                    κάνε τερματισμό του προγράμματος
    τέλος της επιλογής
συνέχεια της επανάληψης
```

Εδώ η εντολή ‘ανάλογα με την επιλογή του χρήστη’ είναι σαν ένα διακόπτη πολλών θέσεων που προκαλεί την ενεργοποίηση της κατάλληλης πράξης από αυτές που είναι διαθέσιμες. Ποια είναι η προσέγγιση στη μοντελοποίηση αυτής της κατάστασης; Αναμφισβήτητα η μοντελοποίηση βασίζεται σε συμβάντα. Προσδιορίζουμε ποιο συμβάν μπορεί να συμβεί και να επηρεάσει το σύστημα και αντιστοιχίζουμε κομμάτια προγράμματος που μπορούν να αντιμετωπίσουν αυτό το συμβάν.

Παραδείγματα κάποιων συμβάντων είναι:

- Ο χρήστης ενεργοποιεί μια επιλογή από το μενού.
- Ο χρήστης δείχνει σε ένα αντικείμενο στην οθόνη και πατάει το πλήκτρο του ποντικιού.
- Ένας διαρρήκτης πέφτει πάνω σε μια φωτεινή δέσμη.
- Ένα μήνυμα φτάνει στην τηλεπικοινωνιακή γραμμή.

Σε ένα σύστημα υπολογιστή με παράθυρα, μενού, πλήκτρα και ποντίκι, μπορεί να υπάρχουν πολλές διαφορετικές επιλογές από τις οποίες ο χρήστης μπορεί να διαλέξει οποιαδήποτε. Αυτή η μοντελοποίηση που είναι βασισμένη στα συμβάντα είναι η τρίτη και όλο και σημαντικότερη προσέγγιση στη δόμηση προγραμμάτων. Κατά πάσα πιθανότητα, η καταλληλότερη μέθοδος σχεδίασης είναι η αντικειμενοστρεφής σχεδίαση.

Ποια η διαφορά ανάμεσα στη μοντελοποίηση βάσει δομής δεδομένων και βάσει συμβάντων; Η πρώτη μπορεί να χρησιμοποιηθεί μόνο όταν η μορφή των δεδομένων είναι καλά καθορισμένη (όπως στο παράδειγμα των αριθμών παραπάνω), ενώ η δεύτερη μπορεί να αντεπεξέλθει και σε μη αναμενόμενες αλληλουχίες δεδομένων εισόδου.

Και στις τρεις μεθόδους σχεδίασης προγράμματος, βάσει δεδομένων, λειτουργιών, ή συμβάντων, η δομή του προγράμματος αντικατοπτρίζει ή προσομοιώνει μια πραγματική κατάσταση.