HY352 : ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ,
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ,
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

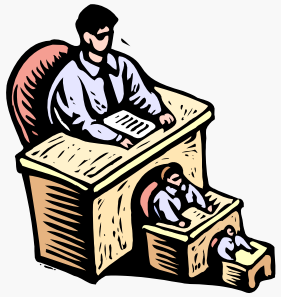
ΔΙΔΑΣΚΩΝ
Αντώνιος Σαββίδης

CSD

ΕΝΟΤΗΤΑ 3

ΣΧΕΔΙΑΣΤΙΚΕΣ ΠΡΟΟΠΤΙΚΕΣ ΚΑΙ ΔΟΜΗΜΕΝΟΣ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΣ

Αριθμός διαλέξεων 2 – Διάλεξη 2η



CSD

Ρόλος των προοπτικών (1/3)

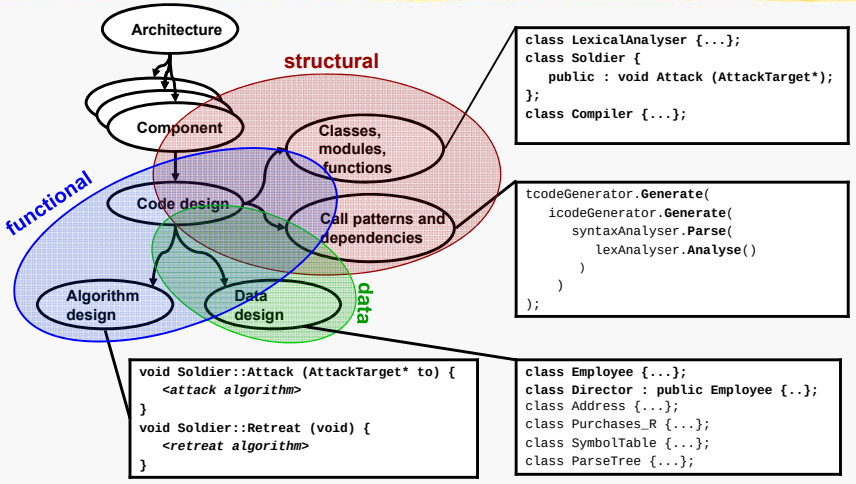
- Η σχεδίαση του λογισμικού δεν μπορεί να αντιμετωπιστεί με μία μόνο τεχνική
- Χρειαζόμαστε εναλλακτικές προοπτικές, όπου η κάθε μία παρουσιάζει μια εξειδικευμένη οπτική γωνία τη σχεδίασης, βοηθώντας στην καλύτερη κατανόηση και οργάνωση του κώδικα υλοποίησης

CSD

Ρόλος των προοπτικών (2/3)

- ***Δεδομένα*** *τα* οποία διαχειρίζεται το λογισμικό
- **Δομή** του κώδικα, με ιεραρχική κατάτμηση, ώστε να λύνει το πρόβλημα που πρέπει να λύνει, καθώς και οι πιθανές εξαρτήσεις κλήσεως μεταξύ των ξεχωριστών τμημάτων
- ***Λειτουργία*** του κώδικα κατά την πραγματική εκτέλεση του προγράμματος, με τη ροή δεδομένων και ελέγχου μεταξύ των λειτουργικών τμημάτων, καθώς και λεπτομερής σχεδίαση αλγορίθμων
- ***Συμπεριφορά*** του συστήματος, τον τρόπο που πρέπει να αντιδρούν τα λειτουργικά τμήματα όταν συγκεκριμένα γεγονότα συμβαίνουν, για επαλήθευση ορθής λειτουργίας

Ρόλος των προοπτικών (3/3)



Περιεχόμενα

- Σχεδιαστικές προοπτικές
  - Ο ρόλος των προοπτικών από άλλη οπτική γωνία
  - *Functional design – λειτουργική σχεδίαση*
  - Behavioral analysis – συμπεριφεριολογική ανάλυση
- Δομημένος προγραμματισμός
  - Βασικές ιδιότητες
  - Συγγενείς μέθοδοι σχεδίασης
  - Σχέση και σύνδεση και με οντοκεντρική σχεδίαση

Functional design (1/2)

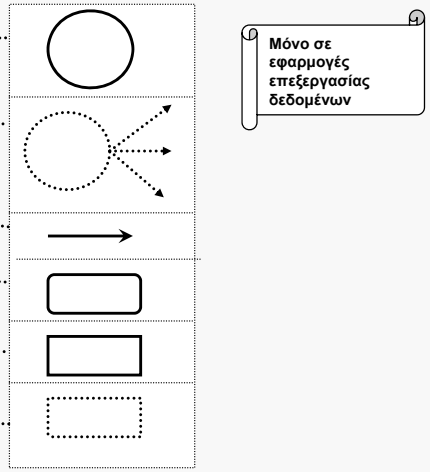
- Τα σχετικά μοντέλα εξάγουν τις βασικές λειτουργίες που επιτελεί το λογισμικό, συμπεριλαμβάνοντας ροή δεδομένων και ροή ελέγχου
- Εμπλέκουν και αναπαριστούν σε ποικίλα επίπεδα λεπτομέρειες σχεδίασης της αλγοριθμικής λογικής

Functional design (2/2)

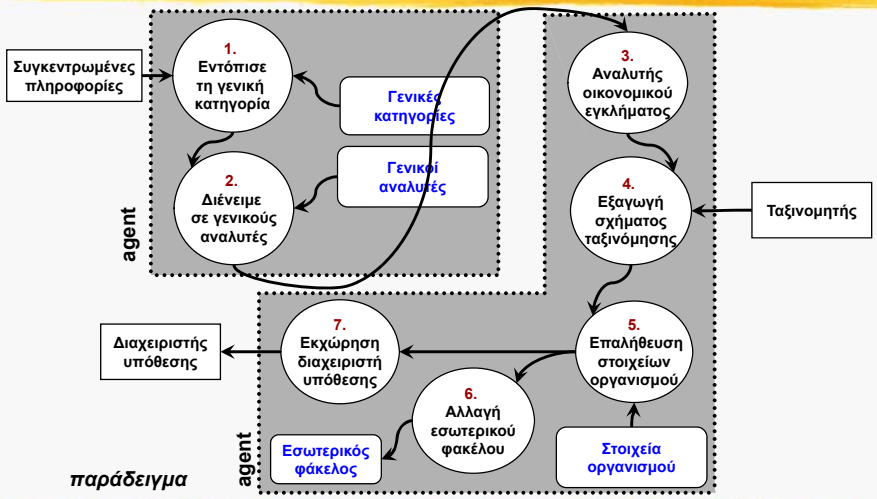
- ***Data Flow Diagrams*** *–DFD, διαγράμματα ροής δεδομένων*. Αντικατοπτρίζουν τις ανάγκες ροής δεδομένων, μοντελοποιώντας τμήματα επεξεργασίας και αποθήκευσης δεδομένων
- ***Flow Charts*** *– FC, διαγράμματα ροής*. Πρόκειται για μία από τις παλαιότερες γραφικές μεθόδους αναπαράστασης αλγοριθμικής λογικής
- ***Pseudo Code****, ψευδοκώδικας*. Η εναλλακτική άποψη στα διαγράμματα ροής, με τη χρήση κειμένου, στη μορφή μη αυστηρά τυποποιημένου κώδικα προγράμματος

Data Flow Diagrams (1/4)

- *Process* }
  - Διεργασία
- *Control flow* }
  - Ροή ελέγχου
- *Data flow* }
  - Ροή δεδομένων
- *Data store* }
  - Αποθήκη δεδομένων
- *External entity* }
  - Εξωτερική οντότητα
- *Grouping* }
  - Ομαδοποίηση



Data Flow Diagrams (2/4)



Data flow diagrams (3/4)

- *Πότε εφαρμόζονται (1/2)*
  - Δείχνουν πως τα δεδομένα εισόδου μετατρέπονται σε αποτελέσματα εξόδου μέσα από μία ακολουθία επεξεργαστικών μετασχηματισμών
  - Ευκολονόητο μοντέλο το οποίο μπορεί να εφαρμοστεί σε περιπτώσεις όπου υπάρχουν σημαντικές επεξεργασίες δεδομένων:
    - τα δεδομένα εισόδου συλλέγονται, διαχειρίζονται, φιλτράρονται και τροποποιούνται, παρέχοντας τα αποτελέσματα στα ενδιαφερόμενα τμήματα για περαιτέρω επεξεργασία
  - Δεν αναπαριστούν ροή ελέγχου, εκτός από περιορισμένης εμβέλειας σχέσεις μέσω άτυπων συνδέσμων ελέγχου

Data flow diagrams (4/4)

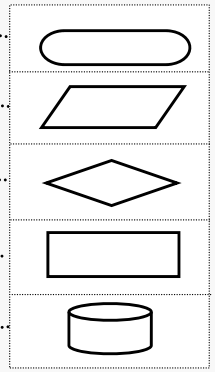
- *Πότε εφαρμόζονται (2/2)*
  - Ανεξάρτητα διαγράμματα μπορούν να ζωγραφιστούν για διαφορετικές ομάδες / φάσεις / τομείς επεξεργασίας δεδομένων
  - Ανεξάρτητα διαγράμματα μπορούν απ' ευθείας να συσχετιστούν με αντίστοιχα λειτουργικά τμήματα η αντικείμενα
  - Η αρίθμηση των διεργασιών εφαρμόζεται για λόγους τεκμηρίωσης και δεν σχετίζεται με την διάταξη των διεργασιών κατά την εκτέλεση
  - Είναι γενικά απλά και εύκολα στη χρήση, χωρίς να εμπλέκουν λεπτομέρειες σχεδίασης δομών δεδομένων
  - Αποτυγχάνουν να αναπαραστήσουν λειτουργική κατάτμηση, καθιστώντας τα ως ***καταλληλότερα για μακροσκοπική ανάλυση ροής δεδομένων***

Flow charts (1/5)

- Τα διαγράμματα ροής παρέχουν ένα λεξιλόγιο γραφικών συμβόλων για την περιγραφή αλγορίθμων
- Είναι μία από τις παλιές καλές μεθόδους για σχεδίαση της λογικής ελέγχου ροής του προγράμματος – program control logic

Flow charts (2/5)

- *Start / end* }
  - Έναρξη / τερματισμός
- *Input / Output* }
  - Είσοδος / έξοδος
- *Conditional* }
  - Υπό συνθήκη εκτέλεση
- *Action* }
  - Εντολή
- *Storage* }
  - Αποθήκευση



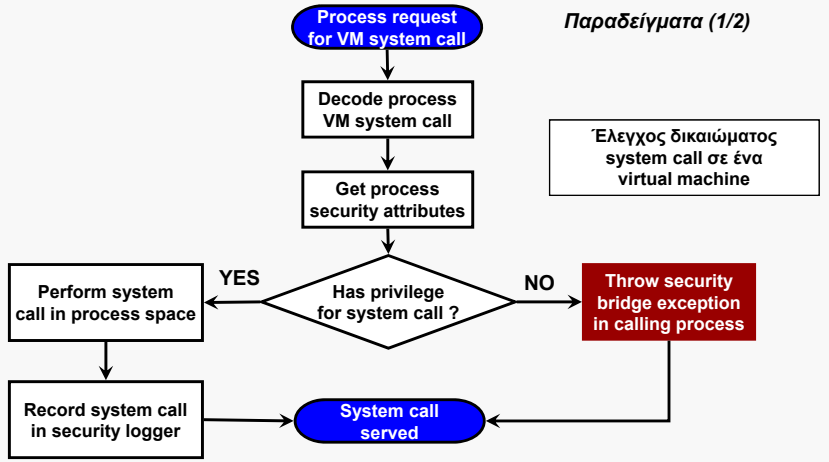
…υπάρχει και πλήθος επιπλέον συμβόλων

Flow charts (3/5)

- *Πότε εφαρμόζονται*
  - Συνήθως δεν συνιστώνται για σχεδίαση κώδικα – ο ψευδοκώδικας θεωρείται ως ευκολότερη μέθοδος
  - Δεν επάγουν αρχιτεκτονική πληροφορία – εμπεριέχουν μόνο υπολογιστικές οντότητες
  - ➔ …αλλά μπορούν να υιοθετηθούν για την ανταλλαγή, εξέταση και τεκμηρίωση ορισμένων είτε κρίσιμων χαρακτηριστικών ροής ελέγχου του συστήματος, η πολύ λεπτομερών παγιωμένων αλγορίθμων

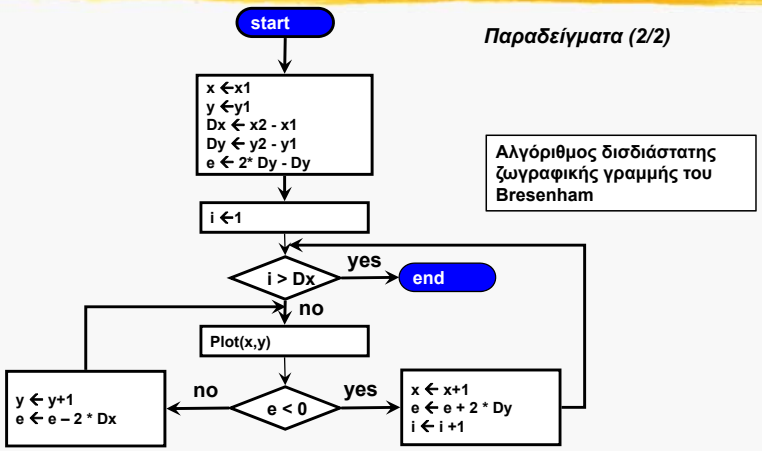
Flow charts (4/5)

Παραδείγματα (1/2)



Flow charts (5/5)

Παραδείγματα (2/2)



Αλγόριθμος δισδιάστατης ζωγραφικής γραμμής του Bresenham

Pseudo code (1/4)

- Πρόκειται για μία ιδιαίτερα διαδεδομένη μέθοδο για τη σχεδίαση κώδικα, με τη χρήση μη αυστηρά τυποποιημένου ρεπερτορίου προγραμματιστικών στοιχείων
  - που όμως είναι πολύ κοντά, τόσο λεκτικά όσο και συντακτικά, σε πραγματικές γλώσσες προγραμματισμού
- Η έμφαση δίνεται στην έκφραση των χαρακτηριστικών ροής ελέγχου και αλγοριθμικής λογικής,
  - αφαιρώντας ευκόλως εννοούμενη τυπολογία
  - επιτρέποντας συνήθως τη χρήση φυσικής γλώσσας για περιγραφή πολύπλοκης λογικής, όταν η τελευταία πρόκειται να αναλυθεί περισσότερο σε κάποια επόμενο «πέρασμα»

Pseudo code (2/4)

- Χαρακτηριστικά χρήσης (1/2)
  - Είναι η πιο κοινή μέθοδος για περιγραφή αλγορίθμων στα αρχικά στάδια σχεδίασης του τρόπου υλοποίησης των συναρτήσεων
  - Οι προγραμματιστές τείνουν να εφαρμόζουν τη δικιά τους μέθοδο, κυρίως δανειζόμενοι στοιχεία από την εκάστοτε γλώσσα προγραμματισμού που χρησιμοποιούν, αναμιγμένη με φυσική γλώσσα και ποικίλους ad hoc συμβολισμούς
  - Είναι πολύ σύνηθες οι προγραμματιστές να μην ακολουθούν με συνέπεια μία συγκεκριμένη τυπολογία, ακόμη και όταν γράφουν ψευδοκώδικα για διαφορετικά τμήματα του ίδιου λογισμικού συστήματος

Pseudo code (3/4)

- *Χαρακτηριστικά χρήσης (2/2)*
  - Ο ψευδοκώδικας εφαρμόζεται σε όλο το σύνολο των προγραμματιστικών στοιχείων, π.χ:
    - *data structure* – δομές δεδομένων
    - *functions declaration* – δήλωση συναρτήσεων
    - *function definition* – ορισμός (υλοποίηση) συναρτήσεων
    - *block* – σύνολο εντολών
    - *statement* - εντολή
    - *expression* – έκφραση
    - *class* – κλάση
    - *variable declaration* – δήλωση μεταβλητών

Pseudo code (4/4)

```
record point { x, y }
function draw line (point start, point end)
begin
  x   = start.x,  y = start.y
  Dx = end.x – start.x, Dy = end.y – start.y
  e   = 2 * Dy – Dx
  for i=1 to Dx do
     begin
       draw(x,y)
       while e > 0 do
           y   = y+1, e = e – 2* Dx
       x = x + 1, e = e + 2 * Dy
     end
end
```

Αλγόριθμος δισδιάστατης ζωγραφικής γραμμής του Bresenham

Πιθανές αλλαγές:

begin	→ {	end	→ {
function	→ *empty*	do	→ *empty*
record	→ *empty*	to	→ -> ή :

παράδειγμα

Περιεχόμενα

- Σχεδιαστικές προοπτικές
  - Ο ρόλος των προοπτικών από άλλη οπτική γωνία
  - Functional design – λειτουργική σχεδίαση
  - *Behavioral analysis – συμπεριφεριολογική ανάλυση*
- Δομημένος προγραμματισμός
  - Βασικές ιδιότητες
  - Συγγενείς μέθοδοι σχεδίασης
  - Σχέση και σύνδεση και με οντοκεντρική σχεδίαση

Behavioral analysis (1/2)

- Τα σχετικά μοντέλα αποκαλύπτουν και αναπαριστούν σχέσεις αιτίου και αποτελέσματος - cause and effect, κυρίως για βασικές κατηγορίες γεγονότων (αιτίες), και των άμεσα συσχετιζόμενων αντιδράσεων (αποτελέσματα) ενός λογισμικού συστήματος
- Ο ορισμός των σχέσεων αιτίου / αιτιατού βοηθά στη γρήγορη επαλήθευση ορθότητας θεμελιωδών συμπεριφεριολογικών χαρακτηριστικών ενός συστήματος, ειδικά εάν ορισμός αυτών των σχέσεων μπορεί να εκφραστεί με τυπικό τρόπο και να επαληθευτεί αυτόματα – formal verification
- Υιοθετούνται για τον ορισμό και την κατανόηση της λειτουργικής συμπεριφοράς αρχιτεκτονικών τμημάτων η του συνολικού συστήματος

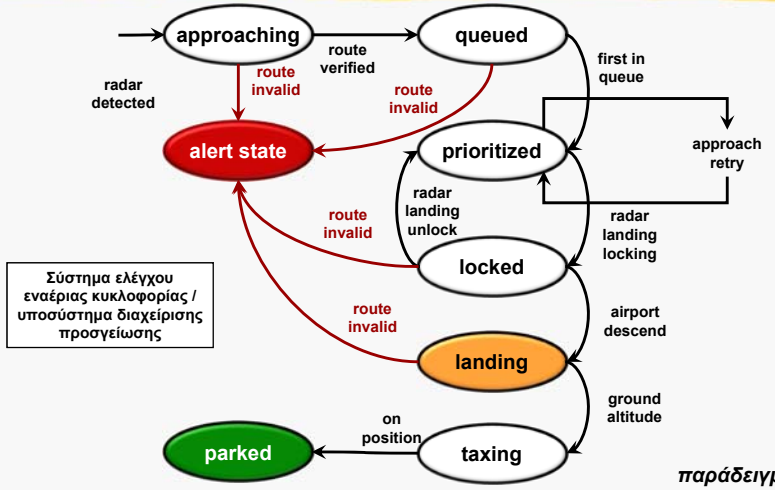
Behavioral analysis (2/2)

- ***State Transition Diagrams*** – *διαγράμματα (μετάβασης) καταστάσεων*. Αυτά αναπαριστούν με γραφικό τρόπο τη λογική του προγράμματος σε μορφή αυτομάτων πεπερασμένων καταστάσεων – finite state automata, εμπλέκοντας καταστάσεις (states), μεταβάσεις (transitions), και ενέργειες (actions)
- ***Cause Effect Tables*** – *πίνακες αιτίου αποτελέσματος*. Πρόκειται κυρίως για πίνακες τεκμηρίωσης με τους οποίους καταγράφονται οι σημαντικότερες κατηγορίες από αίτια / γεγονότα, με τα αντίστοιχα κάθε φορά αποτελέσματα
- ***Formal Specification*** – *τυπική περιγραφή*. Αφορούν τυπικά υπολογιστικά μοντέλα τα οποία υιοθετούνται για την εξαντλητική περιγραφή αλγορίθμων κρίσιμης λογικής προγράμματος, επιτρέποντας την εφαρμογή μεθόδων τυπικής επαλήθευσης

State transition diagrams (1/4)

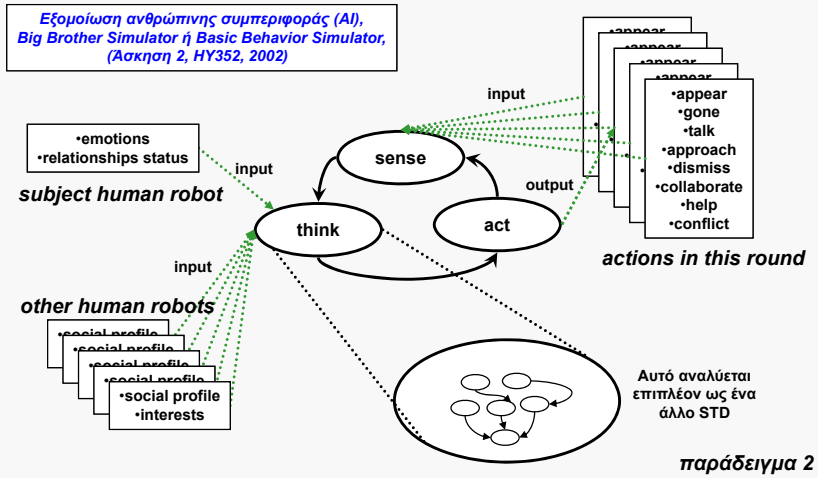
- Μπορούν να εφαρμοστούν όταν καταστάσεις και μεταβάσεις, ή γεγονότα και αντιδράσεις εμπλέκονται απ' ευθείας στην λογική και στη ροή ελέγχου του λογισμικού συστήματος
- Αποτυπώνουν κεντρικές ροές ελέγχου και συνήθους οδηγούν σε βασικά σενάρια λειτουργίας

State transition diagrams (2/4)



παράδειγμα 1

State transition diagrams (3/4)



παράδειγμα 2

State transition diagrams (4/4)

- *Πότε εφαρμόζονται*
  - Η λογική που αναπαρίσταται μέσω διαγραμμάτων καταστάσεων μπορεί εύκολα να μελετηθεί, να αξιολογηθεί και να επαληθευτεί
    - Ωστόσο, αφότου η αρχική σχεδιαστική λογική «υλοποιηθεί» σε κώδικα, είναι συνήθως δύσκολο να μπορεί να εντοπιστεί μέσα στον πηγαίο κώδικα συγκεντρωμένη εξ ολοκλήρου σε «ένα σημείο»
    - Αυτό είναι εφικτό μόνο σε περιπτώσεις που το ίδιο το διάγραμμα αποδίδει λεπτομερώς κάποια αλγοριθμική λογική
  - Χρησιμεύουν ως ελεγκτικά σενάρια, αξιολογώντας κατά πόσο στο πραγματικό σύστημα γίνεται η μετάβαση στις εκάστοτε περιγραφόμενες καταστάσεις, όταν τα αντίστοιχα αναφερόμενα αίτια η γεγονότα λαμβάνουν χώρα
  - Κατασκευαστικά, συνήθως η λογική μετάβασης καταστάσεων για κάθε γεγονός υλοποιείται με μία event-based micro-architecture, μέσα σε ανεξάρτητους, ανά τύπο γεγονότος, επεξεργαστές

Cause Effect Tables (1/3)

- Ο στόχος τους είναι η απαρίθμηση και τεκμηρίωση ορισμένων σημαντικών σχέσεων του τύπου ***αίτιο → αποτέλεσμα*** (cause → effect)
- Αυτή του είδους η τεκμηρίωση συνιστά πολύτιμη και συνάμα ευανάγνωστη πληροφορία σχετικά με τη συμπεριφορά του συστήματος
- Αιτίες είναι:
  - αντιλαμβανόμενα γεγονότα, εσωτερικά η εξωτερικά (events)
  - είσοδος σε συγκεκριμένες καταστάσεις (state reached)
  - αποτίμηση μίας πολύπλοκης συνθήκης σε κάποια συγκεκριμένη τιμή (pre-condition)
- Αποτελέσματα είναι:
  - Ενέργειες που πρέπει να εκτελεστούν
  - Συνθήκες που πρέπει να τηρούνται μετά το αίτιο (post-condition)
  - Γεγονότα που πρέπει να συμβούν

Cause Effect Tables (2/3)

Κατασκευή compiler

Αίτιο / συμβάν	Αποτέλεσμα / αντίδραση
Parse error	Μήνυμα. Τερματισμός.
Expression type checking error	Μήνυμα. Αγνόησε την έκφραση.
Function argument list error	Μήνυμα. Αγνόησε την κλήση.
Re-definition error	Μήνυμα. Αγνόησε τη δήλωση.
Undefined variable error	Μήνυμα. Αγνόησε την έκφραση.

Παράδειγμα 1

Cause Effect Tables (3/3)

Ανάπτυξη περιβάλλοντος Windows

Αίτιο / συμβάν	Αποτέλεσμα / αντίδραση
Window gain focus	Γραφικές ενέργειες. Παρήγαγε σχετικό *gain* event.
Window loose focus	Γραφικές ενέργειες. Παρήγαγε σχετικό *loose* event.
Window resize	Υπολογισμός χώρου περιεχομένων παραθύρων. Καθαρισμός περιεχομένων παραθύρων. Αποστολή *repaint* event στα περιεχόμενα παράθυρα.
Window destroy	Αποστολή *destroy* event στο σχετικό παράθυρο. Καθαρισμός περιοχής παραθύρου. Αποστολή *repaint* event στα επικαλυπτόμενα παράθυρα.

Παράδειγμα 2

Formal specification

- Μέθοδοι τυπικών περιγραφών εφαρμόζονται κυρίως σε:
  - κρίσιμα προς ασφάλεια συστήματα (safety critical systems) και
  - πρωτόκολλα δικτύων (network protocols)
  - → Απαιτούν εξαντλητική περιγραφή των αλγορίθμων, και βασίζονται στη θεωρία των συστημάτων μετάβασης καταστάσεων (state transition systems - STS)
- Οι τεχνικές αυτές ανήκουν στον τομέα της αυτοματοποιημένης επαλήθευσης (computer aided verification),
  - ενώ μπορούν να περιγράψουν ιδιότητες που δεν μπορούν να εξαχθούν από την ανάλυση πεπερασμένης ιστορίας εκτέλεσης (finite trajectories) ενός συστήματος, οι οποίες και λέγονται – *liveness properties*
- Η πιο κλασική περίπτωση ελέγχου μέσω συστημάτων επαλήθευσης είναι η προσβασιμότητα καταστάσεων (state reachability)
  - Μέσω της περιγραφής, κάποιες καταστάσεις ορίζονται ώστε να συνδέονται με συγκεκριμένα σενάρια συμπεριφοράς, όπως λανθασμένη λειτουργία, αντίδραση σε λάθη, η dead locks

Περιεχόμενα

- Σχεδιαστικές προοπτικές
  - Ο ρόλος των προοπτικών από άλλη οπτική γωνία
  - Functional design – λειτουργική σχεδίαση
  - Behavioral analysis – συμπεριφεριολογική ανάλυση
- *Δομημένος προγραμματισμός*
  - Βασικές ιδιότητες
  - Συγγενείς μέθοδοι σχεδίασης
  - Σχέση και σύνδεση και με οντοκεντρική σχεδίαση

Structured design (1/17)

- *Βασικές ιδιότητες*
  - Μία στρατηγική σχεδίασης η οποία βασίζεται στην οργάνωση του συστήματος σε ανεξάρτητα λογικά ***λειτουργικά τμήματα***
  - Αντικατοπτρίζει την σταδιακή κατάτμηση των τμημάτων σε μικρότερα, τα οποία αναπαριστούν δηλωμένη μεν λειτουργικότητα, αλλά όχι ακόμη υλοποιημένη – ***δηλαδή κάτι σαν black boxes***
  - Έμφαση δίνεται στην έννοια του ***modularity*** (*καλή ποιότητα κατάτμησης*) η οποία ορίζεται με μία σειρά από σχεδιαστικές ιδιότητες

Structured design (2/17)

- Ιστορικά συγγενείς σχεδιαστικές μέθοδοι
  - Structured charts και data flow diagrams
- Η σχεδίαση του προγράμματος είναι μία επαναληπτική διαδικασία σταδιακής εξειδίκευσης (step-wise refinement process):
  - Επαναληπτική top-down εξειδίκευση του κώδικα με «αναβαλλόμενη» υλοποίηση συναρτήσεων
    - έως ότου οι πιο αρχέτυπες και απλές συναρτήσεις που χρησιμοποιούνται υλοποιηθούν
  - Ιεραρχικά η εξειδίκευση είναι κατά «πλάτος» (breadth-first specialization)

Structured design (3/17)

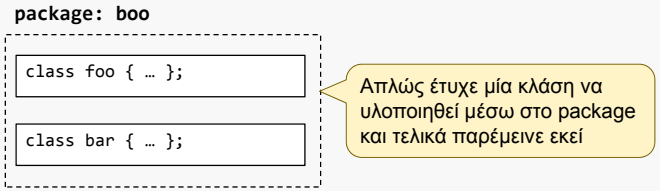
- *Modularity*, *καλή ποιότητα κατάτμησης*
  - Cohesion, *ταίριασμα*
    - Πόσο οι λειτουργίες που έχουν εκχωρηθεί στο ίδιο λειτουργικό τμήμα ταιριάζουν μαζί
      - Θα πρέπει να υλοποιούν μία λειτουργική οντότητα με ένα κοινό στόχο και ρόλο
  - Coupling, *αλληλεξάρτηση*
    - Αποτελεί ένδειξη του βαθμού αλληλεξάρτησης μεταξύ των διαφορετικών λειτουργικών τμημάτων
      - Όσο περισσότερα λειτουργικά τμήματα εξαρτώνται μεταξύ τους, τόσο εντονότερες αλληλεξαρτήσεις υπάρχουν
      - Τα συστήματα χαλαρών αλληλεξαρτήσεων είναι πιο πρόσφορα σε τροποοίηση, συντήρηση, επέκταση και επαναχρησιμοποίηση

Structured design (4/7)

- *Προσοχή στη σύνδεση των ορισμών με τις τεχνικές που ήδη έχουμε μελετήσει*
  - ➔ Η έννοια του ταιριάσματος (cohesion) είναι σε πλήρη συμφωνία με το ορισμό της αρχιτεκτονικής συγγένειας των λειτουργιών ενός τμήματος
    - Βάσει του λειτουργικού ρόλου
  - ➔ Η έννοια της αλληλεξάρτησης αντικατοπτρίζει τις εξαρτήσεις κλήσεων και το πόσο επιρρεπές είναι ένα τμήμα στις αλλαγές άλλων τμημάτων
    - Όσο χαλαρότερες και αραιότερες οι αλληλεξαρτήσεις, τόσο ευκολότερες και ανεξάρτητες οι μεταβολές τοπικής κλίμακας στο λογισμικό

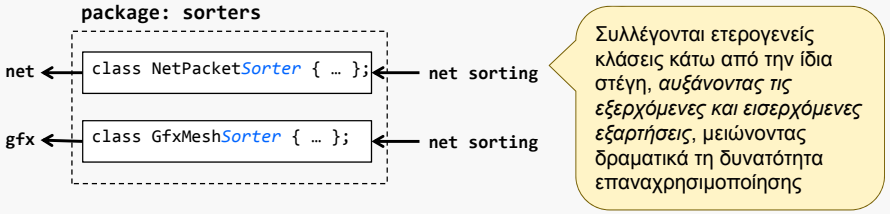
Structured design (5/17)

- *Cohesion (1/8)*
  - **Κατηγορία**: Συμπτωματικό
  - **Σχέση**: Απλώς τυχαία κοινή παρουσία στον κώδικα
  - **Αξία**: Αρνητική, σχεδόν πάντα λάθος



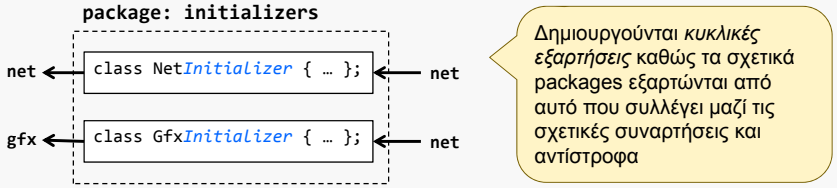
Structured design (6/17)

- *Cohesion (2/8)*
  - **Κατηγορία**: Αλγοριθμικό
  - **Σχέση**: Παρόμοιες λειτουργίες σε χαμηλό επίπεδο
  - **Αξία**: Αρνητική, απλώς παρόμοιες λειτουργίες αλλά με διαφορετικό σκοπό



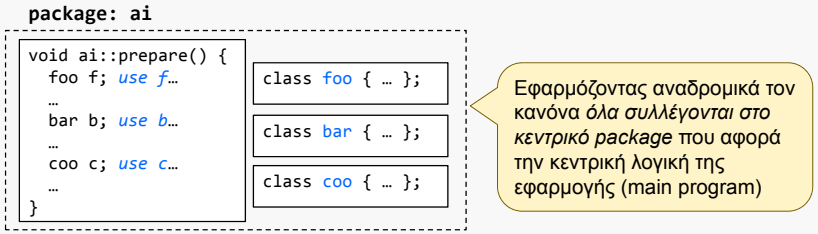
Structured design (7/17)

- *Cohesion (3/8)*
  - **Κατηγορία**: Χρονικό
  - **Σχέση**: Εργάζονται» περίπου στο ίδιο χρονικό διάστημα
  - **Αξία**: Αρνητική, χρονική σύμπτωση



Structured design (8/17)

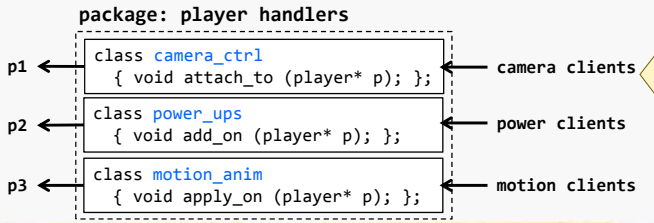
- *Cohesion (4/8)*
  - **Κατηγορία**: Διαδικαστικό
  - **Σχέση**: Στοιχεία της ίδιας λογικής ακολουθίας ενεργειών του προγράμματος
  - **Αξία**: Αρνητική, οδηγεί σε μονόλιθους



Εφαρμόζοντας αναδρομικά τον κανόνα *όλα συλλέγονται στο κεντρικό package* που αφορά την κεντρική λογική της εφαρμογής (main program)

Structured design (9/17)

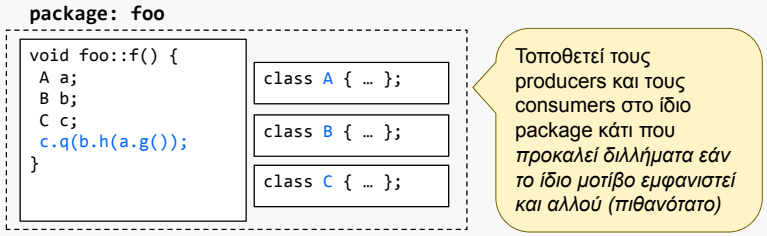
- *Cohesion (5/8)*
  - **Κατηγορία**: Επικοινωνιακό, εισόδου / εξόδου
  - **Σχέση**: Χρησιμοποιούν τα ίδια δεδομένα εισόδου, ή παράγουν την ίδια έξοδο
  - **Αξία**: Αρνητική, επίσης αν συμβαίνουν και τα δύο εφευρίσκουμε πολλές φορές το ίδιο



Εισάγονται αδικαιολόγητες αλληλεξαρτήσεις ενώ *δημιουργεί διλήμματα όταν έχουμε εναλλακτικές εισόδους*

Structured design (10/17)

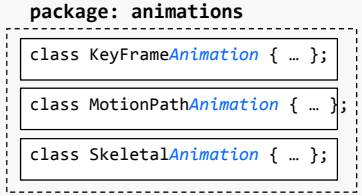
- *Cohesion (6/8)*
  - **Κατηγορία**: Ακολουθιακό
  - **Σχέση**: Το ένα τμήμα τροφοδοτεί την έξοδό του ως είσοδο στο άλλο
  - **Αξία**: Αρνητική, προκαλεί μονόλιθους



Τοποθετεί τους producers και τους consumers στο ίδιο package κάτι που *προκαλεί διλλήματα εάν το ίδιο μοτίβο εμφανιστεί και αλλού (πιθανότατο)*

Structured design (11/17)

- *Cohesion (7/8)*
  - **Κατηγορία**: Λειτουργικό
  - **Σχέση**: Συστατικά ενός κοινού λειτουργικού ρόλου / ευθύνης στο σύστημα
  - **Αξία**: Θετική, ακολουθεί την αρχιτεκτονική συγγένεια



Επειδή χρησιμοποιούμε ονόματα και σημασιολογία *θέλει προσοχή να αποφύγουμε την παγίδα της Αλγοριθμικής κατηγορίας*. Επίσης χρειαζόμαστε την τεχνική αντιστροφής των εξαρτήσεων πολύ συχνά.

Structured design (12/17)

- *Cohesion (8/8)*
  - **Κατηγορία**: Δεδομένων
  - **Σχέση**: Επεξεργάζονται και χρησιμοποιούν τα ίδια δεδομένα
  - **Αξία**: Αρνητική, μονόλιθοι πάνω σε δεδομένα

```
package: icode_handlers
class icode_generator { … };
class icode_optimizer { … };
class icode_analyzer { … };
class icode_validator { … };
```

Η κατηγορία αυτή εμφανίζει την ανάγκη τα δεδομένα να είναι διαθέσιμα σε πολλούς με ένα κοινό API. Πολύ εύκολα δημιουργεί μονόλιθους και διλλήματα, πχ, ένας *icode_generator* χρησιμοποιεί και τον symbol table – που πηγαίνει τότε;

Structured design (13/17)

Αλληλεξάρτηση, Coupling

Κατηγορία	Σχέση μεταξύ των τμημάτων	Σχεδιαστική αξία *to loosen up coupling*
Τροφοδότηση δεδομένων	***Το ένα τμήμα «περνάει» παραμέτρους / δεδομένα στο άλλο μέσω κλήσεως***	***Θετική (the black box concept)***
Χρήση κοινών δεδομένων	**Το ένα τμήμα μοιράζεται κάποια «ξένα» δεδομένα απ' ευθείας με το άλλο**	**Αρνητική, μάλλον πρέπει να ομαδοποιηθούν στο ίδιο τμήμα**
Μεταφορά ελέγχου	**Το ένα τμήμα «οδηγεί» στο άλλο μεταφέροντας τον έλεγχο του προγράμματος σε αυτό**	**Ουδέτερη εάν είναι μονής κατεύθυνσης, αρνητική εάν συμβαίνει εκατέρωθεν**
Αμοιβαία πρόσβαση	**Το ένα τμήμα έχει ελεύθερη πρόσβαση στα περιεχόμενα του άλλου**	**Αρνητική, μετατρέπει τον κώδικα σε κουβάρι. Γίνονται ένα τμήμα, η πιθανότερα σχεδιάζουμε εξ αρχής**

Structured design (14/17)

- ***Αντιστροφή των εξαρτήσεων***
  - ***A→B*** με ένα μικρό υποσύνολο μεθόδων οι οποίες σχετίζονται λειτουργικά μεταξύ τους
  - **A→C** με παρόμοιο τρόπο και καταλήγουμε σε ***A'→C***

```
class SpriteAnimator { Sprite* s; …};   → class Sprite {};
class TextAnimator   { Text* t; …};     → class Text {};
class WindowAnimator{ Window* w; …};    → class Window {};
```

Το παραπάνω μπορεί να επαναληφθεί…

```
struct Animatable {  ***incomplete syntax
 virtual Set_X/Get_x = 0;
 virtual Set_Y/Get_Y = 0;
 virtual Set_W/Get_W = 0;
 virtual Set_H/Get_H = 0;
};
class Animator { Animatable* a; …};
```

```
struct AnimatableSprite :
public Animatable  {
  Sprite* s;
  virtual Get_x (void) const
    { return s->GetPos().x; }
  … και τα υπόλοιπα…
};
```


Structured design (15/17)

- *Σύνδεση με την αντικειμενοστραφή σχεδίαση (1/3)*
  - Στη δομημένη σχεδίαση, τα λειτουργικά τμήματα – modules, συγκεντρώνουν τέτοια λειτουργικότητα ώστε να εξασφαλίζεται καλής ποιότητας κατάτμηση
  - Βάσει των ορισμών του ταιριάσματος και της αλληλεξάρτησης, αυτό συνεπάγεται ότι :
    - *Συναρτήσεις που συνδράμουν στον ίδιο λειτουργικό ρόλο ομαδοποιούνται*
    - *Συναρτήσεις που έχουν αποκλειστική κοινή πρόσβαση σε δεδομένα μάλλον ομαδοποιούνται*
    - *Κανένα τμήμα δεν έχει άμεση πρόσβαση στα περιεχόμενα άλλων τμημάτων*
    - *Τα τμήματα δεν μοιράζονται δεδομένα μεταξύ τους*

Structured design (16/17)

- *Σύνδεση με την αντικειμενοστραφή σχεδίαση (2/3)*
  - Η μέθοδος *αντικειμενοστραφούς* σχεδίασης αποτελεί μία ***ομαλή μετάβαση και εξέλιξη της δομημένης σχεδίασης***
    - ένα βήμα μπροστά στην υποστήριξη υψηλής ποιότητας κατάτμησης, με εξασφάλιση καλού ταιριάσματος και χαμηλών αλληλεξαρτήσεων
  - Στη δομημένη σχεδίαση τα βασικά στοιχεία είναι τα λειτουργικά τμήματα, τα οποία εξυπηρετούν ένα κοινό λειτουργικό στόχο και εμπεριέχουν κοινά δεδομένα. Ωστόσο, κατά την μετάβαση στον κώδικα:
    - Δεν υπάρχει κάποια άμεση προγραμματιστική οντότητα η οποία να αντικατοπτρίζει την έννοια της λειτουργικής μονάδας και της ενθυλάκωσης δεδομένων, εκτός από έμμεσες μεθόδους όπως αυτή του αρχείου και της βιβλιοθήκης (library / package)
    - Η γένεση του αντικειμενοστραφούς προγραμματισμού δίνει υπόσταση σε αυτές τις έννοιες μέσω συγκεκριμένων προγραμματιστικών οντοτήτων

Structured design (17/17)

- *Σύνδεση με την αντικειμενοστραφή σχεδίαση (3/3)*
  - Είναι λάθος η αντίληψη ότι ο αντικειμενοστραφής προγραμματισμός είναι μία σχεδιαστική μέθοδος η οποία εστιάζεται περισσότερο στα δεδομένα, παρά στις λειτουργίες
  - Είναι λάθος να αντιμετωπίζεται ο αντικειμενοστραφής προγραμματισμός σαν μία εντελώς διαφορετική οπτική γωνία σχεδίασης
  - → Ο αντικειμενοστραφής προγραμματισμός είναι η εξέλιξη και βελτίωση της δομημένης σχεδίασης
    - → Τυποποιώντας την έννοια ενός δομημένου τμήματος (***κλάση***)
    - → Τυποποιώντας τις ιδιότητες και τις σχέσεις μεταξύ των δομημένων τμημάτων (***μέλη, εμβέλεια, κανόνες πρόσβασης***)
    - → Τυποποιώντας τις μεθόδους επαναχρησιμοποίησης (***κληρονομικότητα, πρότυπα***)
    - → Παρέχοντας πλήρη προγραμματιστική υποστήριξη (***γλώσσες προγραμματισμού και μοντελοποίησης***)