



Μοντέλα παραγωγής λογισμικού

- ➔ Γιατί υπάρχουν;
- ➔ Σκοπεύουν στην οργάνωση της διαδικασίας παραγωγής σε συγκεκριμένα βήματα και αποφάσεις, με στόχο το τελικό προϊόν λογισμικού, με τον καλύτερο τρόπο

- Water-fall model
- Exploratory model
- Prototyping model
- Spiral model
- Incremental model
- **Inductive model**

HY352, 2010

A. Σαββίδης

Slide 30 / 75



Δημιουργία πρωτοτύπων συστημάτων (1/4)

- Όσο γρηγορότερα και οικονομικότερα προσεγγίσουμε την φάση αξιολόγησης και ελέγχου, τόσο αυξάνουμε την ποιότητα και καταλληλότητα του τελικού συστήματος
 - Σκοπός είναι να πετύχουμε όσο το δυνατόν περισσότερους κύκλους *ανάπτυξης* → *αξιολόγησης* → *τροποποίησης*, με το ίδιο κόστος και στον ίδιο χρόνο
- ➔ Η πρωτότυπη ανάπτυξη ορίζεται ως:
 - ➔ Μερική εξομοίωση εξωτερικής συμπεριφοράς, με απλοποιήσεις εσωτερικής δομής και λειτουργίας

HY352, 2010

A. Σαββίδης

Slide 58 / 75



Περιγραφή απαιτήσεων (3/4)

- Η περιγραφή αυτή περιλαμβάνει δύο βασικές κατηγορίες απαιτήσεων:
 - Πληροφορίες σχετικά με το *τι πρέπει να κάνει το σύστημα*, γνωστές ως λειτουργικές προδιαγραφές (*functional requirements*)
 - Π.χ. αυτόματη ανανέωση κάρτας
 - Πληροφορίες σχετικά με *περιορισμούς και προϋποθέσεις του συστήματος*, γνωστά ως μη-λειτουργικές προδιαγραφές (*non-functional requirements*)
 - Π.χ. ανανέωση το πολύ σε 2 sec

HY352, 2010

A. Σαββίδης

Slide 66 / 75



Ανάλυση απαιτήσεων (1/6)

- Είναι η μελέτη της περιγραφής απαιτήσεων με στόχο την λεπτομερή και σχολαστική καταγραφή των λειτουργικών και μη λειτουργικών προδιαγραφών
 - Η αρχική αυτή ανάλυση, προκαλεί αρκετές τροποποιήσεις, επεξηγήσεις, και επιπλέον προσθήκες στο αρχικό SOR κείμενο
 - Καταγράφονται οι απαιτήσεις του συστήματος σε μία περισσότερο οργανωμένη μορφή από το SOR

HY352, 2010

A. Σαββίδης

Slide 70 / 75



Έλεγχος υλοποίησης και αξιοπιστίας (5/5)

Κατηγορία λαθών	Περιγραφή
Transient - <i>περιστασιακά</i>	Occurs only with certain inputs
Permanent - <i>μόνιμα</i>	Occurs with all inputs
Recoverable – <i>αναρρώσιμα</i>	System recovers itself
Unrecoverable – <i>μη αναρρώσιμα</i>	Operator intervention needed
Non-corrupting – <i>μη καταστροφικά</i>	Data not affected
Corrupting - <i>καταστροφικά</i>	Data get corrupted

HY352, 2010

A. Σαββίδης

Slide 27 / 49



Γρήγορος προσδιορισμός (1/4)

- Ένα συχνό και εύλογο ερώτημα είναι «*πώς αρχίζουμε να μελετάμε και να προσδιορίζουμε την αρχιτεκτονική*»;
 - Συνήθως ο τρόπος χειρισμού είναι «οπτικός», δηλ. πρώτη εμφαση δίνεται στην τοπολογική συνδεσμολογία ως τρόπο αναπαράστασης
 - Η συνδεσμολογία αυτή προσφέρει οπτικό έλεγχο των τμημάτων ενώ αποτυπώνεται εύκολα τα δύο θεμελιώδη χαρακτηριστικά: της *ιεραρχικής κατάταξης* και της *λειτουργικής συνεργίας*
 - Στα πρώτα στάδια η αρχιτεκτονική αποτελείται από βασικά τμήματα (μακροσκοπική αποτύπωση) και είναι αρκετά ρευστή
 - *Η σχεδίαση βασίζεται πάντα γενικών λειτουργικών ρόλων και όχι σε κλάσεις*
 - δε σκεφτόμαστε βάσει κάποιων γλώσσας και ούτε σχεδιάζουμε την υλοποίηση

HY352, 2010

A. Σαββίδης

Slide 14 / 80



Επίπεδα αρχιτεκτονικής (1/3)

- *Macro-architecture*
 - Αποτελεί την συνολική / ευρύτερη αρχιτεκτονική του συστήματος και χαρακτηρίζει δομικά το σύστημα
 - Η εμβέλεια τους περιορίζεται κυρίως σε συγκεκριμένες κάθε φορά κατηγορίες συστημάτων
 - Δεν υπάρχει κριτήριο ως προς το μέγεθος των συστημάτων που αντιπροσωπεύει μία αρχιτεκτονική
 - Κάθε αρχιτέκτονας του λογισμικού πρέπει να γνωρίζει όλες τις σχετικές macro-architectures
 - Μπορεί ακόμη να εμφανιστεί ακόμη και στην ανάλυση ενός συγκεκριμένου τμήματος (δηλ. όχι σε macro επίπεδο)

HY352, 2010

A. Σαββίδης

Slide 21 / 80



Επίπεδα αρχιτεκτονικής (2/3)

- *Micro-architecture*
 - Αποτελούν αρχιτεκτονικές λύσεις για κατηγορίες αρχιτεκτονικών τμημάτων.
 - Μπορεί πολλά στιγμιότυπα μίας micro-architecture να εμφανίζονται, και να υλοποιούνται, σε ένα λογισμικό σύστημα.
 - Κάθε αρχιτέκτονας και σχεδιαστής πρέπει να γνωρίζει όλες τις σχετικές micro-architectures.

HY352, 2010

A. Σαββίδης

Slide 22 / 80



Βασικά αρχιτεκτονικά μοντέλα (1/2)

- Layered architectures
 - Επιπέδων / στρωμάτων
- Sequential architectures
 - Ακολουθιακής επεξεργασίας
- Event-based architectures
 - Βασισμένων σε γεγονότα
- Agent-based architectures
 - Βασισμένων σε λογισμικούς πράκτορες
- Component-based architectures
 - Βασισμένων σε ανεξάρτητα λογισμικά τεμάχια
- Plug-in architectures
 - Βασισμένων σε δυναμικά τεμάχια

HY352, 2010

A. Σαββίδης

Slide 25 / 80



Layered architectures (4/8)

Δομή κώδικα (1/5)

Layer k , top layer. Πιο πρόσφορο για τροποποιήσεις.

Functions	F_k1, \dots, F_kn_k	and data types	T_k1, \dots, T_kn_k	K
Functions	$F_{k-1}1, \dots, F_{k-1}n_{k-1}$	and data types	$T_{k-1}1, \dots, T_{k-1}n_{k-1}$	$K-1$
.....				
Functions	F_21, \dots, F_2n_2	and data types	T_21, \dots, T_2n_2	2
Functions	F_11, \dots, F_1n_1	and data types	T_11, \dots, T_1n_1	1

Layer 1, bottom layer. Συνήθως το πιο επισφαλές σε τροποποιήσεις (η αλλιώς «άβατο»)

- KANONAS. Στην υλοποίηση κώδικα σε οποιοδήποτε layer j , επιτρέπεται να κληθεί μία συνάρτηση F εάν και μόνο εάν ισχύει: $F \in \{F_{j+1}1, \dots, F_{j+1}n_{j+1}\} \cup \{F_j1, \dots, F_jn_j\} \cup \{j\text{-layer inner functions}\}$

HY352, 2010

A. Σαββίδης

Slide 30 / 80



Sequential architectures (6/8)

Δομή κώδικα (1/2)

Φάση επεξεργασίας j

Εξωτερικές εξαρτήσεις	Τύποι δεδομένων εισόδου	Είσοδος j : T_j1, \dots, T_jn_j
	Input Accessor API	Είσοδος j : A_j1, \dots, A_jn_j
Εσωτερική υλοποίηση	Εσωτερική υλοποίηση	F_j1, \dots, F_jn_j
	Τύποι δεδομένων εξόδου	Εξόδος j : $T_j^*1, \dots, T_j^*n_j$
	Output Accessor API	Εξόδος j : $A_j^*1, \dots, A_j^*n_j$

- KANONAS. Στην υλοποίηση κώδικα επιτρέπεται να κληθεί μία συνάρτηση F εάν και μόνο εάν ισχύει: $F \in \{A_j1, \dots, A_jn_j\} \cup \{F_j1, \dots, F_jn_j\} \cup \{A_j^*1, \dots, A_j^*n_j\}$. Δεν επιτρέπεται να υπάρχουν κλήσεις συναρτήσεων που ανήκουν σε άλλες φάσεις

HY352, 2010

A. Σαββίδης

Slide 42 / 80



Περιεχόμενα

- Σχεδίαση λογισμικού: αρχές και μέθοδοι
 - Αρχές σχεδίασης λογισμικού (σύντομη εισαγωγή)
 - Σχεδιαστικές προοπτικές
 - Data modeling – μοντελοποίηση δεδομένων
 - Structural design – δομική σχεδίαση
 - Functional design – λειτουργική σχεδίαση
 - Behavioral analysis – συμπεριφερσιολογική ανάλυση

HY352, 2010

A. Σαββίδης

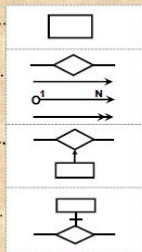
Slide 3 / 43



E/R diagrams (2/5)

- Entity types
 - τύποι οντοτήτων
- Relationships
 - Σχέσεις
- Associative entity type
 - Συσχετιστική οντότητα
- Supertype / subtype
 -

Σύμβολα



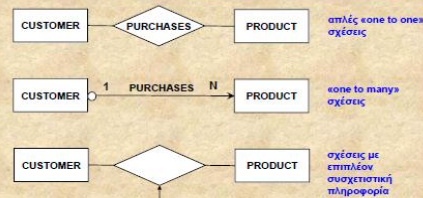
HY352, 2010

A. Σαββίδης

Slide 13 / 43



E/R diagrams (3/5)



Παραδείγματα

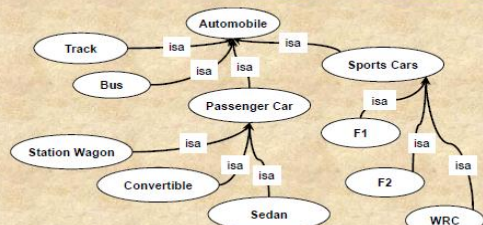
HY352, 2010

A. Σαββίδης

Slide 14 / 43



Object diagrams (3/10)



Παραδείγματα

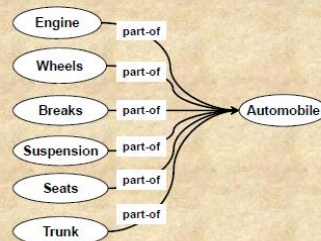
HY352, 2010

A. Σαββίδης

Slide 19 / 43



Object diagrams (4/10)

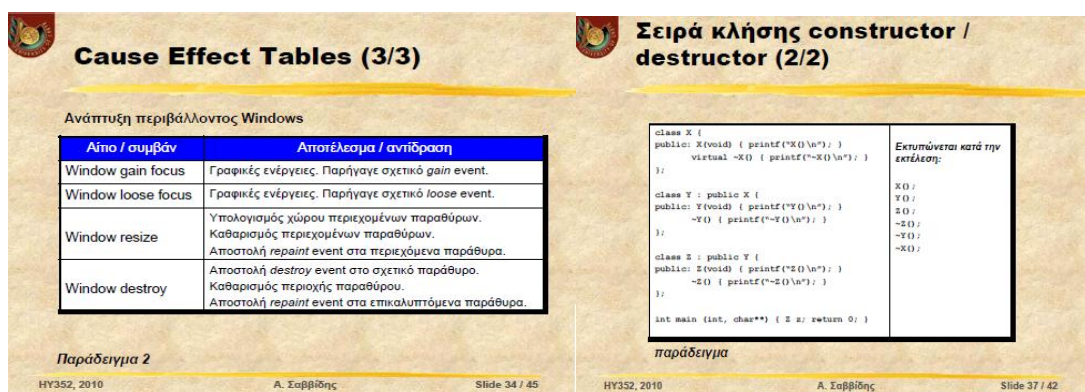
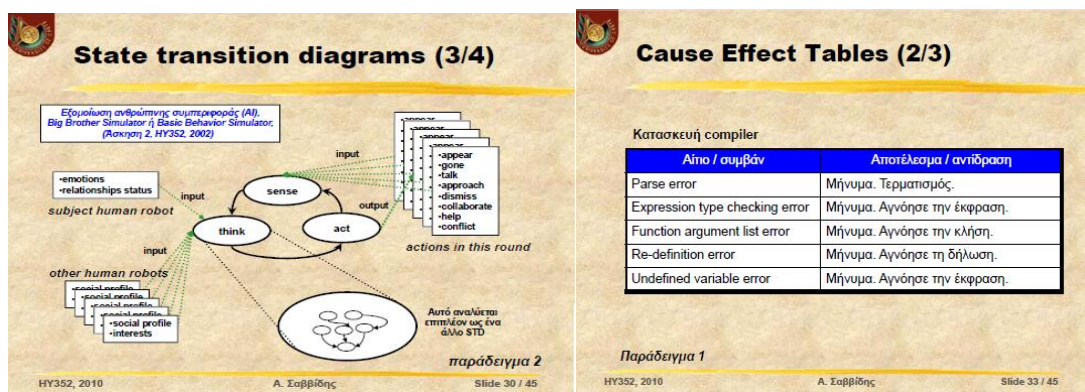
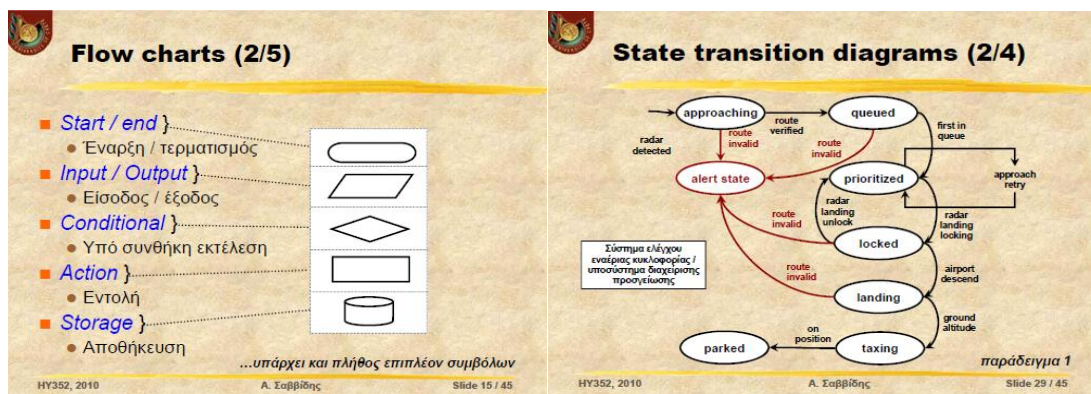
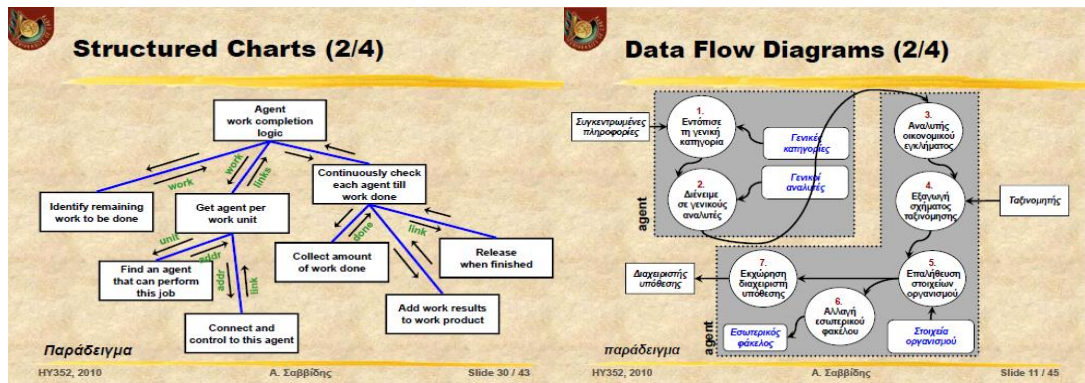


Παραδείγματα

HY352, 2010

A. Σαββίδης

Slide 20 / 43



Κατηγορίες constructor (1/11)

- **Empty (default)**
 - Κενός
- **Parameterized**
 - Παραμετροποιημένος
- **Copy (copy)**
 - Αντιγραφέας
- **Converter**
 - Μετατροπέας
- **Decoder**
 - Αποκωδικοποιητής

Εισαγωγή – design by contract (4/5)

- **Βασικά στοιχεία του design by contract**
 - Βασίζεται σε τρία είδη assertions, με κάθε assertion να ορίζεται είτε ως μια boolean έκφραση ή, εάν είναι πολύπλοκη, μέσω μιας συνάρτησης που επιστρέφει bool:
 - ◆ **Preconditions**
 - συνθήκες που καθορίζουν κατά την εκτέλεση τη νομιμότητα κλήσης μελών - ορίζονται για κάθε συνάρτηση μέλος
 - ◆ **Postconditions**
 - συνθήκες που που καθορίζουν την ορθή περάτωση κλήσης μελών - ορίζονται για κάθε συνάρτηση μέλος
 - ◆ **Invariants**
 - αλλιώς τα αξιώματα της κλάσης, που είναι συνθήκες οι οποίες ορίζουν την ορθότητα και νομιμότητα των στιγμιότυπων μιας κλάσης

Εισαγωγή – design by contract (5/5)

- Το παράδειγμα πάνω στο οποίο θα μελετήσουμε το design by contract είναι η αφηρημένη κλάση `stack`, όπως ορίζεται μερικώς παρακάτω:

```
class Stack {
    FUNCTIONS:
        bool    empty (void);
        void    push (type);
        type    top (void);
        void    pop (void);
        int     total (void);
        bool    full (void);
        Stack (void);
        ~Stack();
};
```

Preconditions (2/2)

παράδειγμα

```
class Stack {
  PRECONDITIONS:
  pop:  not FUNCTIONS.empty()
  pop:  not FUNCTIONS.empty()
  push: not FUNCTIONS.full()
};

if (stackInstance.PRECONDITIONS.pop())
  stackInstance.FUNCTIONS.pop();
else
  Κατάλληλες ενέργειες ανάλογα με την
  κατάσταση περιτήτων, λογική του προγράμματος;
```

Υλοποίηση των preconditions των συναρτήσεων - μέλη

Έλεγχος preconditions στα τμήματα clients

Το πρόγραμμα είναι ετοιμο για τον επόμενο

Postconditions (2/2)

παράδειγμα

```
class Stack {
  POSTCONDITIONS:
    pop():      total equals old_total-1 and
                 not FUNCTIONS.full();
    push(x):     total equals old_total+1 and
                 not FUNCTIONS.empty() and
                 x equals FUNCTIONS.top();

  FUNCTIONS:
    push(x) {
      old_total = total;
      Ασυνήθι - ασυνήθως push του στοιχείου x.
      assert(POSTCONDITIONS.push(x)); Ελέγχος postcondition
    }
    στη νέα κλάση
};
```

Invariants (3/3)

παράδειγμα

```

class Stack {
  INVARIANTS:
    axiom():      0 ≤ total ≤ MAX_ELEMS and
                  total equals 0 or not FUNCTIONS.empty() and
                  total equals MAX_ELEMS or not FUNCTIONS.full();

  FUNCTIONS:
    push(x) {
      Ομως πριν...
      assert(INVARIANTS.axiom());
    }
    pop() {
      Ομως πριν...
      assert(INVARIANTS.axiom());
    }
}

```

*Ελεγχος invariant
πριν την κλήση σε
κάθε μέθοδο*

Κατηγορίες προτύπων (1/5)

- Εάν υπάρχει κάποια κατηγοριοποίηση στο βιβλίο «Design Patterns», δεν είναι πάντα ξεκάθαρη η αντιστοίχιση ορισμένων προτύπων σε συγκεκριμένη κατηγορία
- Τέτοιου είδους σχήματα ταξινόμησης θα πρέπει να θεωρούνται ως ανοικτά:
 - κάθε νέο πρότυπο δεν πρέπει απαραίτητα να ανήκει σε μία υπάρχουσα κατηγορία,
 - αλλά ούτε και θα πρέπει να ορίζει από μόνο του μία νέα κατηγορία

Κατηγορίες προτύπων (2/5)

- **Μερικές ενδεικτικές κατηγορίες (1/2):**
 - **Constructional, creational**
 - Τρόποι παραγωγής στιγμιότυπων
 - **Communicational**
 - Τρόποι επικοινωνίας μεταξύ τμημάτων
 - **Control, coordination**
 - Τρόποι διαχείρισης και διεύθυνσης άλλων τμημάτων
 - **Compositional**
 - Τρόποι σύνθεσης τμημάτων για σύνθετες λειτουργίες
 - **Search**
 - Τρόποι αναζήτησης



Κατηγορίες προτύπων (3/5)

- Μερικές ενδεικτικές κατηγορίες (2/2):
 - **Browsing**
 - Τρόποι διερεύνησης και επίσκεψης πολύπλοκων δομών
 - **Memory**
 - Έξυπνοι τρόποι διαχείρισης της μνήμης
 - **Correctness**
 - Τρόποι πιστοποίησης και ελέγχου λειτουργικής ορθότητας
 - **Optimizers**
 - Τρόποι βελτιστοποίησης απόδοσης
 - **Sharing**
 - Ειδικοί τρόποι κοινής χρήσης εσωτερικών δεδομένων
 - **Transformers**
 - Τρόποι μετάλλξεής τμημάτων χωρίς να επηρεάζονται τα αυθεντικά


HY352, 2010 Α. Σαββίδης Slide 25 / 27



Κατηγορίες προτύπων (4/5)

- Ποια θα μελετήσουμε (1/2)
 - **Browsing**
 - Iterator
 - **Constructional**
 - Factory
 - Prototype
 - **Sharing**
 - Singleton
 - State
 - **Control / coordination / communication**
 - Proxy
 - Dispatch table
 - Black board


HY352, 2010 Α. Σαββίδης Slide 26 / 27



Κατηγορίες προτύπων (5/5)

- Ποια θα μελετήσουμε (2/2)
 - Listener
 - Progress monitoring
 - Undo / redo
 - **Transformers**
 - Adapter
 - View
 - **Memory**
 - Flyweight
 - **Compositional**
 - Decorator


HY352, 2010 Α. Σαββίδης Slide 27 / 27



Αποτυχία πόρων (3/4)

- Μία αποτυχία πόρων μπορεί να συμβεί ενώ ο έλεγχος βρίσκεται στη μέση κάποιας εσωτερικής επεξεργασίας. Για να αποφευχθεί το πρόγραμμα θα πρέπει να:
 - επιστρέψει από αρκετές συναρτήσεις
 - απελευθερώσει όσους πόρους παραχωρήθηκαν ενδιάμεσα
 - να μεταδώσει τον κωδικό και την πληροφορία της αποτυχίας
 - να περιέλθει σε μία τέτοια κατάσταση στην οποία η λειτουργία που απέτυχε να έχει πλήρως ακυρωθεί
- Όταν συναρτήσεις εμπλέκονται σε τέτοιες καταστάσεις θα πρέπει να υλοποιούνται με τρόπο που όλες οι περιπτώσεις λαθών ελέγχονται με κατάλληλες συνθήκες ενώ τα λάθη μεταδίδονται στον caller είτε με τη μορφή επιστρεφόμενων τιμών (παλαιός τρόπος;) η μέσω διαχείρισης exceptions (νέος τρόπος;)


HY352, 2010 Α. Σαββίδης Slide 10 / 44



Προγραμματιστικό σφάλμα (3/6)

- Χρονική απόσταση σφάλματος – *bug time distance*
 - Ορίζεται ως ο χρόνος που μεσολαβεί κατά την εκτέλεση από την γέννηση του σφάλματος, έως το σημείο που γίνεται αντιληπτή η ύπαρξή του
 - Όσο μεγαλύτερος είναι αυτός ο χρόνος, τόσο δυσκολότερος είναι ο εντοπισμός και προσδιορισμός της πραγματικής αιτίας


HY352, 2010 Α. Σαββίδης Slide 15 / 44



Προγραμματιστικό σφάλμα (4/6)

- Χωρική απόσταση σφάλματος – *bug source distance*
 - Ορίζεται άμεσα ως η «απόσταση» μεταξύ του σημείου του κώδικα στο οποίο γεννιέται το σφάλμα, και το σημείο στο οποίο γίνεται για πρώτη φορά αντιληπτό – εκεί που χτυπάει το σφάλμα
 - Αυτή η μετρική δεν έχει ιδιαίτερη αξία πέραν του ότι χρησιμοποιείται σε διαφωνίες μεταξύ των προγραμματιστών για το ποιος ευθύνεται για ένα bug το οποίο μόλις βγήκε στην επιφάνεια:
 - Ενώ το σύμπτωμα εμφανίζεται σε ένα σημείο, δεν είναι απαραίτητο να φταίει ο προγραμματιστής που υλοποίησε αυτόν τον κώδικα
 - Δεν χρειάζεται εφραυχασμός όταν το bug εμφανίζεται σε «αρκιτή απόσταση» από τον κώδικά σας
 - Η απόσταση μπορεί να ορίζεται ως μεγαλύτερη καθώς προχωράμε σε: συνεχόμενες εντολές, blocks, συναρτήσεις, τμήματα, υποσυστήματα.


HY352, 2010 Α. Σαββίδης Slide 16 / 44



Αυτοέλεγχος προγράμματος (4/8)

- Τα assertions ελέγχουν συνθήκες οι οποίες πρέπει να είναι πάντα true, αλλιώς,
 - ή κάποιο σφάλμα έχει ήδη γεννηθεί,
 - ή ένα σφάλμα θα προκληθεί από τις εντολές που ακολουθούν το assertion
- Τα assertions πρέπει να χρησιμοποιούνται μόνο για σφάλματα, και όχι για τις φυσιολογικά αναμενόμενες αποτυχίες πόρων (αυτές θα πρέπει να εντοπίζονται και να διαχειρίζονται κατάλληλα).
- Επίσης τοποθετούμε assertions σε σημεία του κώδικα που δεν αναμένουμε ποτέ να εκτελεστούν, ως *φύλακες της ροής ελέγχου - control flow guards*.

HY352, 2010 Α. Σαββίδης Slide 29 / 44



Iterator (1/7)

- **Πρόβλημα**
 - Παρέχουμε σειριακούς τρόπους πρόσβασης στα περιεχόμενα σύνθετων συλλογών στοιχείων C (container classes) μη απαραίτητα γραμμικά δομημένων, χωρίς να εκπίεται η εσωτερική τους αναπαράσταση
- **Λύση** (μία από τις διάφορες που υπάρχουν)
 - Παρέχουμε έναν αφηρημένο τύπο iterator (ADT) ο οποίος και υλοποιείται πλήρως μέσα στις κλάσεις C, ο οποίος και προσφέρει όλες τις απαραίτητες συναρτήσεις για την πρόσβαση στα περιεχόμενα στοιχεία. Ο αφηρημένος τύπος iterator δεν πρέπει να εμπεριέχει συναρτήσεις οι οποίες βασίζονται σε κάποια συγκεκριμένη υλοποίηση των κλάσεων C.
- **Επιπτώσεις**
 - Διαφορετικοί αλγόριθμοι πρόσβασης από διαφορετικές κλάσεις iterator
 - Το API της κάθε C κλάσης απλουστεύεται
 - Πολλάπλες παράλληλες προσβάσεις είναι εφικτές (ένας iterator διατηρεί την κατάσταση πρόσβασης σε ειδικές τοπικές μεταβλητές)
 - Η κλάσεις C παράγουν / παρέχουν τα στιγμιότυπα / τύπους iterator

HY352, 2010 Α. Σαββίδης Slide 4 / 40

Visitor (1/5)

- Πρόβλημα**
 - Χρειάζεται να εφαρμόσουμε κάποιες λειτουργίες στα στοιχεία μίας συλλογής ή ενός σύνθετου αντικείμενου όταν τα στοιχεία του είναι διαφορετικών τύπων (αλλά γνωστών στην υλοποίηση του σύνθετου αντικείμενου)
- Λύση**
 - Η υλοποίηση της συλλογής παρέχει μία αφηρημένη κλάση *Visitor* με μεθόδους για την επίσκεψη κάθε διαφορετικού συστατικού στοιχείου της συλλογής (τα ανώματά των μεθόδων ταιριάζουν με τους τύπους των στοιχείων). Παρέχεται μία μέθοδος *accept (Visitor)* από τη συλλογή.
 - Όλα τα στοιχεία κληρονομούν από έναν κοινό τύπο. Η σειρά επίσκεψης μπορεί να είναι καλά ορισμένη ή όχι αλλά πάντα τεκμηριώνεται τι ισχύει.
- Επιπτώσεις**
 - Ο client υλοποιεί μία κατάλληλη subclass του *Visitor* και καλεί την *accept* σε ένα κατάλληλο instance της συλλογής
 - Μπορούν να υλοποιηθούν όσες διαφορετικές κλάσεις από *visitors* επιθυμούμε.
 - Δεν απαιτείται η τροποποίηση της συλλογής εάν θέλουμε να εφαρμόσουμε κάποιες επιπλέον λειτουργίες στα συστατικά στοιχεία.

Factory (1/7)

- Πρόβλημα**
 - Το σύστημά μας κατασκευάζεται πάνω από εναλλακτικές παρόμοιες βιβλιοθήκες μέσω των οποίων δημιουργεί στιγμιότυπα διαφορετικών κλάσεων. Θέλουμε να μην εμφανίζονται στον κώδικα εξάρτηση από κάποια τέτοια οικογένεια κλάσεων, με δυνατότητα χρήσης όποιας επιθυμούμε σε διαφορετικές εκδόσεις του συστήματος.
- Λύση**
 - Ενοποίησε τις διάφορες κλάσεις της κάθε οικογένειας κάτω από μία οικογένεια αφηρημένων κλάσεων, έπειτα όρισε ένα αφηρημένο εργοστάσιο (*factory*) στιγμιότυπων, και έπειτα υλοποίησε τα εξειδικευμένα ανά οικογένεια *factories*.
- Επιπτώσεις**
 - Ο κώδικας χρήσης μπορεί να διαλέγει μεταξύ των εναλλακτικών *factories*, καθιστώντας τον εφαρμόσιμο σε διαφορετικές οικογένειες κλάσεων απ' ευθείας.
 - Μπορούν να επεκταθούν οι οικογένειες χωρίς να επηρεάζεται ο αρχικός κώδικας.

HY352, 2010
A. Σαββίδης
Slide 12 / 40
HY352, 2010
A. Σαββίδης
Slide 18 / 40

Prototype (1/2)

- Πρόβλημα**
 - Χρειάζεται να δημιουργήσουμε ακριβή αντίγραφα στιγμιότυπων κάποιας συγκεκριμένης κατάστασης, παρά να δημιουργούμε στιγμιότυπα και να τα φέρομε στην επιθυμητή κατάσταση με κλήσεις μελών. Η κατάσταση αυτή μπορεί να μην είναι πάντα γνωστή κατά την κατασκευή του συστήματος (compile-time), αλλά να αποφασίζεται αλγοριθμικά κατά την λειτουργία (run-time).
- Λύση**
 - Οι αντίστοιχες κλάσεις παρέχουν έναν αντιγράφεα (π.χ. *Clone()*). Τα πρωτότυπα είναι στιγμιότυπα είτε της αυθεντικής κλάσης, ή ειδικά κατασκευασμένης κληρονομίας, εάν η αυθεντική κλάση δεν περιέχει αντιγράφεα και επίσης είναι αδύνατο να τροποποιηθεί.
- Επιπτώσεις**
 - Τα πρωτότυπα μας σώνουν από αρκετό κώδικα, ειδικά εάν η προσέγγιση της επιθυμητής κατάστασης στιγμιότυπου απαιτεί αρκετές και πολύπλοκες κλήσεις. Επίσης, ελαφρύνεται ο προγραμματιστής από την απομνημόνευση όλων αυτών των μελών που θα εμπλεκόνταν μόνο σε τέτοιες κλήσεις.

Singleton (1/4)

- Πρόβλημα**
 - Θέλουμε να επιβάλουμε την ύπαρξη ενός μοναδικού στιγμιότυπου μίας κλάσης, το οποίο είναι πάντα διαθέσιμο όταν το χρειάζεται το πρόγραμμά μας.
- Λύσεις**
 - Όρισε την κλάση με *private constructor* και με ένα τοπικό *private / protected static* στιγμιότυπο της ίδιας της κλάσης.
 - Κάνε μία *lightweight* κλάση μόνο με *static* μέλη και *private static* τοπικά δεδομένα, και με ειδικές *Initialise()* και *CleanUp()* συναρτήσεις
- Επιπτώσεις**
 - Η πρώτη λύση δεν έχει μεν καλό στυλ κλήσεων, αλλά επιτρέπει κληρονομικότητα.
 - Η δεύτερη λύση έχει πολύ καλό στυλ κλήσεων, αλλά δεν επιτρέπει κληρονομικότητα.

HY352, 2010
A. Σαββίδης
Slide 26 / 40
HY352, 2010
A. Σαββίδης
Slide 29 / 40

State (1/7)

- Πρόβλημα**
 - Τα στιγμιότυπα πρέπει να αλλάζουν δραστικά την συμπεριφορά τους, χωρίς ωστόσο να σημαίνει αυτό αλλαγή του API, ανάλογα με διαφορετικές τιμές των μεταβλητών κατάστασης, πρακτικά απαιτώντας λειτουργικές διαφοροποιήσεις οι ο οποίες δεν ταιριάζουν καλά μέσα στην ίδια την κλάση
- Λύση**
 - Ενσωμάτωσε τις λειτουργικές διαφορές σε εναλλακτικές κλάσεις, όλες ως κληρονομίες του ίδιου αφηρημένου API. Δημιούργησε τοπικά αντίστοιχα στιγμιότυπα, και δήλωσε ένα δείκτη στο αφηρημένο API. Όταν η κατάσταση αλλάξει, ο δείκτης αυτός εκχωρείται τη διεύθυνση του αντίστοιχου στιγμιότυπου. Η αρχική κλάση φραγώνει την κλήση των μελών μόνο μέσω του δείκτη στο αφηρημένο API (*late binding*).
- Επιπτώσεις**
 - Πρέπει να οριστούν οι αντίστοιχες κλάσεις ανά κατάσταση, οποίες και περιέχουν όλα τα δεδομένα σχετικά με την κατάσταση. Η ταχύτητα είναι καλύτερη και πάντα σταθερή (*late binding*), ενώ η επέκταση των καταστάσεων δεν αλλάζει τον αρχικό κώδικα (λίγες γραμμές μόνο).

Structured design (3/9)

- Modularity, καλή ποιότητα κατάτμησης**
 - Cohesion, ταιρίασμα**
 - Πόσο καλά τα περιεχόμενα του ίδιου λειτουργικού τμήματος ταιριάζουν μαζί
 - Τα περιεχόμενα κάθε λειτουργικού τμήματος πρέπει να υλοποιούν μία μοναδική λογική 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 συντήρηση, επέκταση και επαναχρησιμοποίηση

HY352, 2010
A. Σαββίδης
Slide 34 / 40
HY352, 2010
A. Σαββίδης
Slide 39 / 46

Structured design (7/9)

- Σύνδεση με την οντοκεντρική 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

Κληρονομιά τύπων / δεδομένων (3/5)

```

class Agent {
    char* name;
    char* hostRunning;
    Agent* controllerAgent;
    ...
}

class Arbitrator : public Agent {
    Agent* agentsArbitrated;
    unsigned totalArbitrated;
    ArbitrationPolicy policy;
    ...
}
        
```

Agent στιγμιότυπο

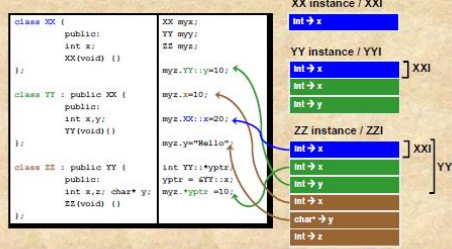
char* → name
char* → hostRunning
Agent* → controllerAgent

Arbitrator στιγμιότυπο

char* → name
char* → hostRunning
Agent* → controllerAgent
Agent* → agentsArbitrated
unsigned → totalArbitrated
ArbitrationPolicy → policy

HY352, 2010
A. Σαββίδης
Slide 43 / 45
HY352, 2010
A. Σαββίδης
Slide 17 / 42

Κληρονομιά τύπων / δεδομένων (5/5)



HY352, 2010

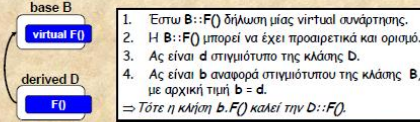
A. Σαββίδης

Slide 19 / 42

Δυναμική αντιστοίχιση (1/7)

- Η χρήση αφηρημένων κλάσεων και κληρονομικότητας είναι στενά συνδεδεμένη με την έννοια της δυναμικής αντιστοίχισης (late binding)

- Στην πραγματικότητα, χωρίς την υποστήριξη δυναμικής αντιστοίχισης, η προγραμματιστική αξία των αφηρημένων κλάσεων είναι μηδαμινή



- Έστω $B::F()$ δήλωση μίας virtual συνάρτησης.
- Η $B::F()$ μπορεί να έχει προαιρετικά και ορισμό.
- Ας είναι d στιγμιότυπο της κλάσης D .
- Ας είναι b αναφορά στιγμιότυπου της κλάσης B , με αρχική τιμή $b = d$.

⇒ Τότε η κλήση $b.F()$ καλεί την $D::F()$.

HY352, 2010

A. Σαββίδης

Slide 17 / 40

Εντροπία λογισμικού (2/7)

Κρίσιμες ερωτήσεις

- Ποιο είναι το ποσοστό του προστιθέμενου πηγαίου κώδικα, επί της αρχικής καλά σχεδιασμένης μάζας, με το οποίο αρχίζει να «θολώνει» η σχεδιαστική εικόνα?
- Ποιες είναι οι συγκεκριμένες περιπτώσεις στις οποίες ο επιπλέον κώδικας αυξάνει την εντροπία?
- Πότε ποσοτικά μπορούμε να πούμε ότι προσεγγίζεται η κρίσιμη μάζα του πηγαίου κώδικα?
- Πώς μπορούμε να ελέγχουμε εάν κινούμαστε σε πορεία αύξησης της εντροπίας ώστε να μπορέσουμε να αντιδράσουμε?

→ Καλή μεταφορά για τα λογισμικά συστήματα είναι η αντιστοίχια με την αύξηση της εντροπίας σε μία πόλη όταν γίνεται άναρχη δόμηση και επέκταση εκτός του αυθεντικού σχεδίου

HY352, 2010

A. Σαββίδης

Slide 5 / 50

Εντροπία λογισμικού (3/7)

Πιθανές απαντήσεις?

- Δεν υπάρχει σχετικό τυποποιημένο μαθηματικό μοντέλο το οποίο να δίνει εξισώσεις αποτίμησης και πρόβλεψης της συνάρτησης αύξησης με καλά ορισμένες παραμέτρους.
- Ωστόσο γνωρίζουμε ότι η αύξηση της επηρεάζεται θετικά από συγκεκριμένες ενέργειες και τακτικές ανάπτυξης.
- Η δυνατότητα των προγραμματιστών να χειρίζονται και να κατασκευάζουν συστήματα μεγάλης κλίμακας είναι άμεσα εξαρτημένη από την δεξιοτήτά τους να αντιμετωπίζουν τη λογισμική εντροπία.

- Συνήθως οι προγραμματιστές αγνοούν ότι η ανάπτυξη συστημάτων δεκαπλάσιου μεγέθους απαιτεί «εκατονταπλάσιες» γνώσεις και δεξιότητες.

HY352, 2010

A. Σαββίδης

Slide 6 / 50

Εντροπία λογισμικού (4/7)

Είναι η «ρίζα του κακού»

- Όποτε προσθέτουμε ένα νέο χαρακτηριστικό σε ένα υπάρχον σύστημα, ουσιαστικά χτίζουμε πάνω στην αυθεντική του σχεδίαση.
- Αυτή η ενέργεια δεν έπεται μίας προσεκτικής αξιολόγησης βασισμένη στην *αντοχή της αυθεντικής σχεδίασης*.
- Ο χρόνος απλώς καταναλώνεται για αλγοριθμική αντιμετώπιση του εκάστοτε νέου προβλήματος / χαρακτηριστικού.

→ αλλά δεν αφιερώνεται χρόνος για την εξασφάλιση της ομαλής και βέλτιστης ενσωμάτωσης στην υπάρχουσα βάση πηγαίου κώδικα

HY352, 2010

A. Σαββίδης

Slide 7 / 50

Δημιουργική αναδιάρθρωση (2/7)

Τι ρόλο παίζει το refactoring (1/2)

- Μία τεχνική εξέλιξης της σχεδίασης, χωρίς ωστόσο να συνιστά σχεδιαστική τεχνική.
- Επεκτείνει τη σχεδίαση πάντα ένα βήμα μακρύτερα από το σημείο στο οποίο θα σταματήσουν οι ενδεχόμενες προσθήκες πηγαίου κώδικα.

- ενώ είναι περισσότερο μία «θεραπευτική» τεχνική, υποθέτει ότι μάλλον κάτι παρόμοιο θα ξαναγίνει και προετοιμάζει «προληπτικά» την σχεδίαση

HY352, 2010

A. Σαββίδης

Slide 50 / 50

Δημιουργική αναδιάρθρωση (3/7)

Τι ρόλο παίζει το refactoring (2/2)

- Σπρώχνει στα όρια την σχεδίαση, στο μέγιστο ανεκτό μέγεθος κώδικα.
- όμως εάν αδυνατούμε πια να αντιμετωπίσουμε την εντροπία μέσω refactoring, ξέρουμε ότι φτάσαμε στο επικίνδυνο σημείο της «μη επιστροφής», όπου η σχεδίαση πρέπει επαναπροσδιοριστεί.
- Δεν πρέπει να θεωρείται ως μία μέθοδος δυναμικής σχεδίασης, ούτε ως μία τακτική επιβίωσης λογισμικών συστημάτων με «λγότερο από άριστη» σχεδίαση, αλλά ως *μέθοδος για βέλτιστη διατήρηση μίας βέλτιστης σχεδίασης*.

- Εάν επιμένετε στην εξέλιξη λάθος σχεδίασης, το re-factoring θα βοηθήσει το λογισμικό σας να ζήσει μεν περισσότερο, αλλά θα οδηγήσει αναμφίβολα σε μία πολύ επιβλητική και άδοξη κατάρρευση

HY352, 2010

A. Σαββίδης

Slide 13 / 50