



Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής & Τηλεπικοινωνιών

Τεχνολογία Λογισμικού

8ο Εξάμηνο 2022-23

Σχεδιαστικές αρχές λογισμικού

Δρ. Κώστας Σαΐδης (saiko@di.uoa.gr)

Ανάλυση vs Σχεδιασμός

- Κατά την ανάλυση ασχολούμαστε με το **τι** θα κάνει το λογισμικό (ποια είναι η επιθυμητή συμπεριφορά;).
- Στη σχεδίαση, η προσοχή μεταφέρεται στο **πώς** θα το κάνει (πώς θα επιτύχουμε την επιθυμητή συμπεριφορά;).

Σχεδίαση λογισμικού

Θα πρέπει να εντοπίσουμε:

- Ποια συστατικά αποτελούν την εφαρμογή μας.
- Πώς αυτά συσχετίζονται μεταξύ τους.
- Πώς διατάσσονται.
- Ποια είναι η εσωτερική τους δομή.

Οι απαντήσεις στα ερωτήματα αυτά, αποτελούν το **σχέδιο του λογισμικού (software design)**.

The most difficult design task is to find the most appropriate decomposition of the whole into a module hierarchy, minimizing function and code duplications.

N. Wirth

Κι η αρχιτεκτονική του λογισμικού;

Περί τίνος πρόκειται

Η λήψη των θεμελιωδών δομικών και σχεδιαστικών αποφάσεων για το λογισμικό που είναι δύσκολο (ακριβό) να αλλάξουν αφού υλοποιηθούν.

Συνοδεύει:

- Ανάλυση απαιτήσεων
- Σχεδιασμό

Σχεδιασμός λογισμικού

Ένα σύνθετο πρόβλημα:

- Ποιος είναι ο καλύτερος τρόπος μετάβασης από τις προδιαγραφές στο σχέδιο του λογισμικού;
- Με πόσα σχέδια μπορεί να υλοποιηθεί κάθε προδιαγραφή και ποιο είναι το «καλύτερο»;
- Πόσο λεπτομερής είναι μια «καλή» αποτύπωση του σχεδίου και τι περιλαμβάνει αυτή;
- Πώς διασφαλίζεται η ποιότητα του λογισμικού;

Ποιοτικά ζητούμενα σχεδιασμού

- Performance - Scalability
- Maintainability - Extensibility
- Security - Safety
- Robustness - Fault-tolerance
- Usability - Reliability

Σας θυμίζουν κάτι;

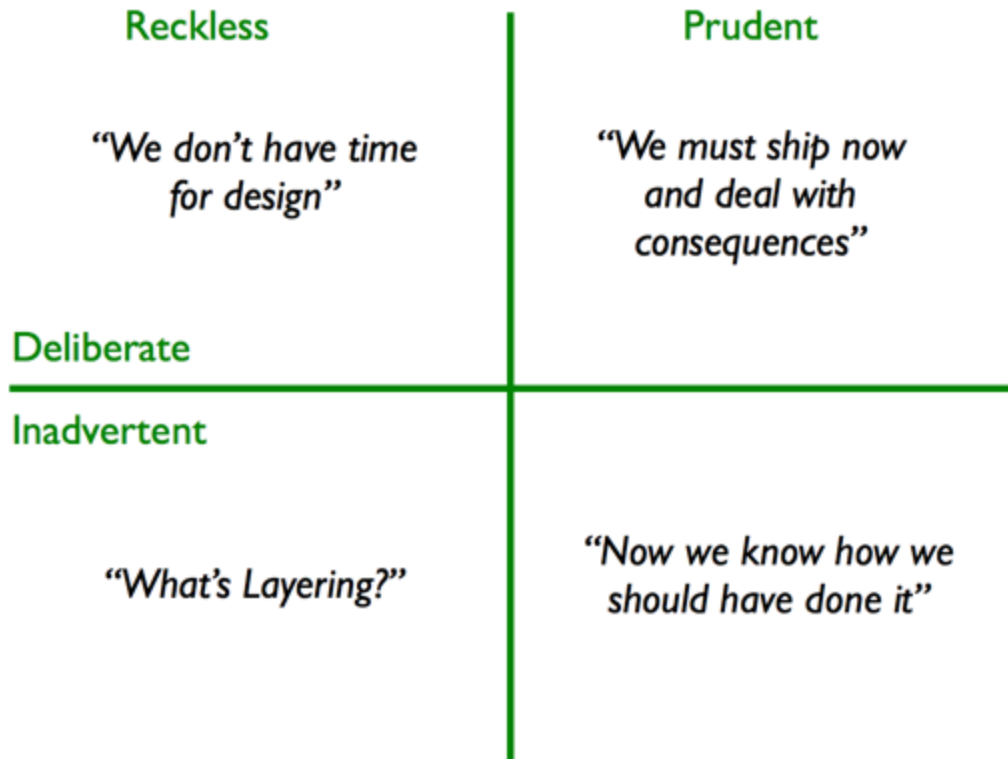
Σχεδιασμός = Συμβιβασμός

- Συνήθως δεν είναι εφικτό να γίνουν όλα καλά.
- Επιλογή των επιθυμητών trade-offs.

Τεχνικό χρέος

- Το κόστος της πρόσθετης προσπάθειας / δουλειάς που θα απαιτηθεί
- αν επιλέξουμε μια "εύκολη" και "γρήγορη" λύση (quick & dirty)
- αντί για τη συνολικά καλύτερη λύση
- στο κάθε σχεδιαστικό πρόβλημα.

Τέσσερα είδη



By Martin Fowler

Το κλασικότερο trade-off

Efficiency vs Abstraction

Programmers have spent far too much time worrying about efficiency in the wrong places at the wrong times; premature optimization is the root of all evil.

Donald Knuth

Αφαίρεση (Abstraction)

Θεμελιώδης έννοια

- "Η εννοιολογική διαδικασία κατά την οποία προκύπτουν γενικοί κανόνες από την εξέταση επιμέρους παραδειγμάτων." (Wikipedia)
- Χρησιμοποιείται σε πολλές επιστήμες.
- Αποτελεί το κύριο εργαλείο σχεδιασμού.

Αφαίρεση

Αναπαράσταση ενός ουσιώδους χαρακτηριστικού του λογισμικού χωρίς τις δευτερεύουσες λεπτομέρειες.

Παράδειγμα

```
interface Item {  
    String getId()  
    void setId(String id)  
    ...  
}  
  
interface Datastore {  
    Item load(String id)  
    void save(Item item)  
}
```

Βελτίωση (Refinement)

- Συμπληρωματική διαδικασία της αφαίρεσης
- Τμηματική προσαρμογή των αφαιρέσεων σε νέες απαιτήσεις, περιορισμούς ή αποφάσεις

Παράδειγμα

```
interface Item {  
    ...  
}  
  
interface Datastore {  
    void load(String id, Consumer<Item> itemConsumer)  
    void save(Item item, Consumer<Result> resultConsumer)  
}  
  
interface Consumer<T> {  
    accept(T t);  
}  
  
interface Result {  
    Item forItem();  
    boolean succeeded();  
    Throwable error();  
}
```


Refactoring

- Η διαδικασία επαναπροσαρμογής του κώδικα ως τμήμα κάποιας βελτιωτικής απόφασης
- Αυτοματοποιείται από πολλά IDEs
- Θα δούμε λεπτομερές παράδειγμα σε επόμενη διάλεξη

Τμηματοποίηση (modularity)

- Αναλύουμε ένα πολύπλοκο σύστημα σε επιμέρους απλούστερα τμήματα
- Σχεδιάζουμε ξεχωριστά το ένα τμήμα από το άλλο
- Συνθέτουμε ξανά τα τμήματα και τις αλληλεπιδράσεις τους σε ένα ενιαίο σύνολο
- Με σκοπό τη διευκόλυνση της υλοποίησης, της συντήρησης και της εξέλιξης του λογισμικού

Απόκρυψη πληροφορίας (information hiding)

- Η απόκρυψη των χαρακτηριστικών που μπορεί να αλλάξουν στο λογισμικό (σχεδιαστικών αποφάσεων, λεπτομερειών υλοποίησης) από τα άλλα τμήματα, ώστε να ελαχιστοποιηθούν οι αλλαγές που απαιτούνται αν/όταν προκύψει η αλλαγή
- Παροχή σταθερών interfaces (αφαιρέσεων) για την επικοινωνία μεταξύ των τμημάτων

Συναφείς έννοιες / αρχές

- Ενθυλάκωση (encapsulation)
- Διαχωρισμός ενδιαφερόντων (separation of concerns)
- Σύζευξη & συνεκτικότητα (coupling & cohesion)

Ενθυλάκωση (encapsulation)

- Διαχωρισμός της δομής από τη συμπεριφορά ενός συστατικού
- Προστασία ενός συστατικού από τη μετάβασή του σε μη έγκυρη κατάσταση
- Μπορεί να θεωρηθεί ως τεχνική υλοποίησης της αρχής της απόκρυψης πληροφορίας

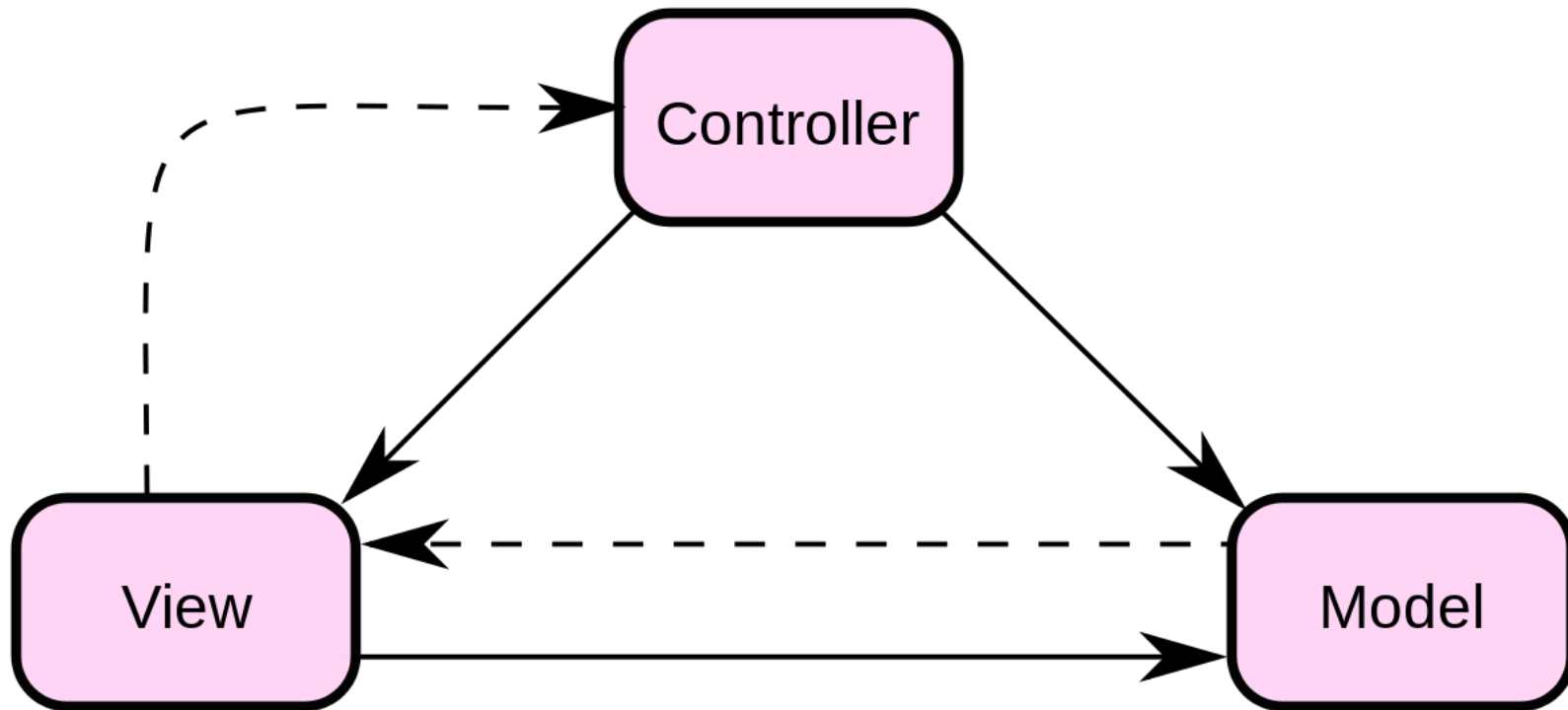
Παράδειγμα

```
class MySQLDatastore implements Datastore {
    private Connection con //encapsulation
    public boolean exists(String id) {
        ResultSet rs = con.execute("select 1 from items where item = $", id)
        return !rs.isEmpty()
    }
    ...
}
```

Επιμερισμός αρμοδιοτήτων (separation of concerns)

- Κάθε τμήμα του λογισμικού επικεντρώνεται στην επίλυση ενός ξεχωριστού ζητήματος (concern)
- Αρχιτεκτονικά επίπεδα (επίπεδο παρουσίασης, επίπεδο επιχειρησιακής λογικής, επίπεδο πρόσβασης δεδομένων)

Παράδειγμα



Θα το κουβεντιάσουμε αναλυτικά σε επόμενη διάλεξη

Σύζευξη & Συνεκτικότητα

Σύζευξη (coupling)

Ο βαθμός της αλληλεξάρτησης μεταξύ δύο συστατικών (κλάσεων, modules, κλπ)

Συνεκτικότητα (cohesion)

Ο βαθμός με τον οποίο τα στοιχεία που ανήκουν σε ένα συστατικό σχετίζονται λειτουργικά (ορθώς συνανήκουν στο ίδιο συστατικό)

Ζητούμενο

Χαλαρή σύζευξη (loose coupling) μεταξύ των συστατικών

Υψηλή συνεκτικότητα (high cohesion) "μέσα" σε κάθε συστατικό

Επαναχρησιμοποίηση (reuse)

- Το λογισμικό δεν πρέπει να ανακαλύπτει κάθε φορά εκ νέου τον τροχό
- Οι καλές αφαιρέσεις και τα καλώς διαχωρισμένα συστατικά είναι εύκολο να επαναχρησιμοποιηθούν

Παράδειγμα

Map/Reduce pattern

```
var animals = ["dog", "cat", "fish"]
var len = function(s) { return s.length; }
var sum = function(a, b) { return a+b; }
animals.map(len).reduce(sum, 0); //10
```

```
var records = //list of person records;
var getAge = function(person) { return currentYear - person.birthYear; }
records.map(getAge).reduce(sum, 0) / records.length; //avg age of people
```

Πρόσθετες σχεδιαστικές αρχές (design principles)

Πώς να δομήσουμε και να υλοποιήσουμε τις αφαιρέσεις
(abstractions)

Μην επαναλαμβάνεσαι (Don't Repeat Yourself)

"Every piece of knowledge must have a single, unambiguous, authoritative representation within a system."

A. Thomas, D. Hunt

Μόνο μια φορά (Once and Only Once)

- Αρχή του Extreme Programming (XP)
- Each and every declaration of behavior should appear Once and Only Once.

DRY vs WET

- WET: We Enjoy Typing
- WET: Waste Everybody's Time

Εφαρμογή της αρχής DRY στην πράξη

- Η επανάληψη (duplication) αυξάνει την τυχαία/τεχνητή πολυπλοκότητα του συστήματος (accidental complexity).
- Οπουδήποτε στον κύκλο ζωής του λογισμικού υπάρχουν "χειροκίνητες" διαδικασίες, θα πρέπει να αυτοματοποιούνται (π.χ. μέσω build system, scripting, κ.ά.).
- Αν προκύπτουν επαναλήψεις στη λογική, κάπου απαιτείται η προσθήκη μιας νέας αφαίρεσης.

Σύνολο αρχών S.O.L.I.D.

- Single responsibility (S)
- Open/closed (O)
- Liskov substitution (L)
- Interface segregation (I)
- Dependency inversion (D)

Μοναδική ευθύνη (Single responsibility)

- Κάθε κλάση/συστατικό πρέπει να έχει μία και μοναδική ευθύνη.
- A class should have only one reason to change.

Ανοικτό/κλειστό (Open/closed)

- Κάθε κλάση/συστατικό πρέπει να είναι ανοικτή σε επεκτάσεις (π.χ. προσθήκη νέων πεδίων ή μεθόδων).
- Κάθε κλάση/συστατικό πρέπει να είναι κλειστή και οριοθετημένη (έτοιμη προς χρήση από τρίτα συστατικά).

Δυνατότητα αντικατάστασης (Liskov substitution)

- Αν το **S** είναι υποτύπος του **T** τότε όλα τα αντικείμενα του δεύτερου θα πρέπει να μπορούν να αντικατασταθούν με αντικείμενα του πρώτου χωρίς να αλλοιωθεί κανένα από τα επιθυμητά χαρακτηριστικά του συστήματος.
- Strong behavioral subtyping.

Επιμερισμός διεπαφών (Interface segregation)

- Κανένα συστατικό δεν πρέπει να εξαρτάται από μεθόδους που δε χρησιμοποιεί.
- Χρήση μικρών και διαφορετικών interfaces για τη θέσπιση των διεπαφών μεταξύ συστατικών.

Ανιστροφή εξαρτήσεων (Dependency inversion)

- Τα υψηλού επιπέδου συστατικά δεν πρέπει να εξαρτώνται από τα χαμηλού επιπέδου συστατικά. Και τα δύο θα πρέπει να εξαρτώνται από κοινές αφαιρέσεις.
- Οι αφαιρέσεις δεν πρέπει να εξαρτώνται από λεπτομέρειες. Οι λεπτομέρειες θα πρέπει να εξαρτώνται από τις αφαιρέσεις.

Παράδειγμα

```
class DataAccessLayer {  
    //private MySQLDatastore store //bad - strong coupling  
    private Datastore store //good  
    DataAccessLayer() {  
        //store = new MySQLDatastore() //bad - strong coupling  
        store = InjectConfig.get('Datastore') //good - "injection"  
    }  
    ...  
}
```

Dependency inversion through injection (απλουστευμένο παράδειγμα)