03 Ανάλυση Απαιτήσεων και Σχεδιασμός Λογισμικού

Τεχνολογία Λογισμικού

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο

Χειμερινό εξάμηνο 2017-18

Δρ. Κώστας Σαΐδης (saiko@di.uoa.gr)

Περιεχόμενα

1. Οι απαιτήσεις λογισμικού και τα είδη τους
2. Από την ανάλυση απαιτήσεων στη σύνταξη των προδιαγραφών και ο ρόλος του επικεφαλής μηχανικού (Software Architect)
3. Μοντελοποίηση απαιτήσεων / προδιαγραφών & σχεδιασμού
4. Σχεδιαστικές αρχές λογισμικού
5. Η έννοια του τεχνικού χρέους (technical/design/code debt)

1. Ανάλυση απαιτήσεων

Απαιτήσεις λογισμικού

Συνήθως είναι ανεξάρτητες από την τεχνολογία υλοποίησης.

- Απαίτηση: Τι θέλουμε να κάνει το λογισμικό
- Ικανοποίηση απαίτησης: Πώς το λογισμικό καλύπτει την απαίτηση (σχεδιαστική απόφαση και υλοποίηση).

Εκτός, βέβαια, αν η υποστήριξη κάποιας τεχνολογίας είναι ρητή απαίτηση

Είδη απαιτήσεων

- Λειτουργικές απαιτήσεις
- Μη λειτουργικές απαιτήσεις
- Απαιτήσεις συστήματος
- Αναδυόμενες απαιτήσεις

Λειτουργικές απαιτήσεις

- Οι απαιτήσεις της λειτουργίας του λογισμικού (συνήθως η νέα λειτουργικότητα που πρέπει να αναπτυχθεί στο πλαίσιο του έργου)
- Παραδείγματα: διαπίστευση και δικαιοδοσία χρηστών, επιχειρησιακή λογική, περιπτώσεις και σενάρια χρήσης, διαχειριστικές λειτουργίες

Διαπίστευση χρηστών (user authentication)

- Σημεία εισόδου στο σύστημα (entry-points) - μηχανισμοί διαπίστευσης χρηστών
- Επιβεβαίωση ότι ο χρήστης είναι:
  - "υπαρκτός" (π.χ. ο λογαριασμός του είναι εγγεγραμμένος στον "πίνακα" των χρηστών της ΒΔ)
  - "έγκυρος" (π.χ. γνωρίζει το προσωπικό του "μυστικό" που έχει συνδεθεί με το λογαριασμό του)
  - "ενεργός" (π.χ. δεν έχει απενεργοποιηθεί από το διαχειριστή)

Δικαιοδοσία χρηστών (user authorization)

- Διαχείριση δικαιωμάτων: ποιος χρήστης επιτρέπεται να εκτελέσει ποια ενέργεια.
- Στατική ή δυναμική ανάθεση/ανάκληση δικαιωμάτων;
- Ρόλοι
- Access Control Lists (ACLs)
- Guards

Επιχειρησιακή λογική (business logic)

- Οι επιχειρησιακές διαδικασίες (business processes)
- Οι επιχειρησιακοί κανόνες (business rules)
- Μερική ή πλήρης "ηλεκτρονικοποίηση" των λειτουργιών του οργανισμού

Πιο απλά

- Τα υποστηριζόμενα σενάρια και περιπτώσεις χρήσης (use cases) του λογισμικού
- Οι ενέργειες, λειτουργίες και διαδικασίες που εκτελούν οι χρήστες μέσω του συστήματος

Διαχειριστικές λειτουργίες (administrative functions)

- Ποιες είναι οι διαχειριστικές λειτουργίες;
- Πώς εκτελούνται;
- Θα παραμετροποιούνται από το χρήστη και πόσο;

Λειτουργίες αναφορών (reporting requirements)

- Τι αναφορές απαιτούνται ;
- Πόσο παραμετροποιήσιμες πρέπει να είναι;
- Ποιος έχει δικαίωμα να τις προσπελαύνει;

Μη λειτουργικές απαιτήσεις

- Απαιτήσεις για ποιοτικά χαρακτηριστικά του λογισμικού
  - Διαθεσιμότητα - Ανάνηψη από καταστροφές
  - Ασφάλεια - Ακεραιότητα
  - Ευελιξία - Επεκτασιμότητα
  - Απόδοση - Απροκρισιμότητα
  - Υποστήριξη διεθνών προτύπων
- Καθορίζουν σε μεγάλο βαθμό την επιτυχία του έργου

Απαιτήσεις συστήματος

- Απαιτήσεις σχετικές με το "σύστημα" ή το "περιβάλλον" στο οποίο εντάσσεται το υπό ανάπτυξη λογισμικό (π.χ. μπορεί να είναι συστατικό ενός μεγαλύτερου όλου)
- Απαιτήσεις υλικού, απαιτήσεις δικτύου, κ.ά

Αναδυόμενες απαιτήσεις

- Ποιοτικά χαρακτηριστικά που προκύπτουν από την ένταξη του λογισιμικού στο "σύστημα" ή "περιβάλλον" και αναφέρονται στη λειτουργία του ως όλον.

Στην πράξη

- Στην πράξη, κάποιες απαιτήσεις μπορεί να βρίσκονται "κάπου ανάμεσα" στα παραπάνω είδη
- Αλλά αυτό ποικίλει ανάλογα με το υπό ανάπτυξη λογισμικό, το περιβάλλον λειτουργίας του και τους εμπλεκόμενους φορείς - χρήστες του
- Ας δούμε μερικά παραδείγματα

Τήρηση ημερολογίων (audit tracking)

- Ποιες ενέργεις καταγράφονται σε ημερολόγια;
- Σε ποιο βαθμό λεπτομέρειας;
- Για ποιο σκοπό και με ποια τελική χρήση;

Μαζική εισαγωγή / εξαγωγή δεδομένων (Data import / export)

- Απατείται η μαζική εισαγωγή / εξαγωγή δεδομένων;
- Θα παραμετροποιείται από το χρήστη και πόσο;
- Ποιοι μορφότυποι αρχείων / δεδομένων θα πρέπει να υποστηρίζονται;

Υποστήριξη διεθνούς προτύπου

- Ανάλογα με την περίπτωση, μπορεί να επηρεάζει:
  - Ένα συγκεκριμένο συστατικό
  - Ένα υποσύνολο των συστατικών
  - Το σχεδιασμό συνολικά

Διαλειτουργικότητα (interoperability)

- Το υπό ανάπτυξη σύστημα θα διαλειτουργεί με τρίτα συστήματα;
- Τι είδους δεδομένα θα ανταλλάσονται;
- Με ποιους μηχανισμούς, μορφότυπους και πρωτόκολλα;

Νομικές ή κανονιστικές απαιτήσεις

- Το υπό ανάπτυξη σύστημα θα πρέπει να είναι σύννομο (προσωπικά δεδομένα, πνευματικά δικαιώματα, κτλ.)
- Το υπό ανάπτυξη σύστημα μπορεί να διέπεται από συγκεκριμένο νομικό ή κανονιστικό πλαίσιο (για παράδειγμα, πληροφοριακά συστήματα δημοσίου τομέα).

Ορισμός των απαιτήσεων

- Σαφήνεια (η απαίτηση είναι σαφώς διατυπωμένη)
- Συνέπεια (η μία απαίτηση δεν αντιβαίνει με την άλλη)
- Πληρότητα (η απαίτηση καλύπτει όλες τις περιπτώσεις)
- Επαληθευσιμότητα (η απαίτηση μπορεί να ελεγχθεί σε σχέση με την υλοποίηση)

Επίσης

Σε μεγάλα έργα

- Καταγωγή/γενεαλογία απαίτησης
  - Ποιος τη ζήτησε
  - Ποιον επιχειρησιακό στόχο εξυπηρετεί

Για παράδειγμα, θα πρέπει να γνωρίζει η ομάδα του έργου αν "το ζητάει ο διευθυντής"

Έγγραφο ανάλυσης απαιτήσεων

Περιγραφή σε φυσική γλώσσα:

- Σκοπός του συστήματος
- Κατηγορίες χρηστών (users & stakeholders)
- Εμβέλεια του συστήματος
- Περιορισμοί
- Παραδοχές
- Απαιτήσεις (λειτουργικές ή μη)

2. Από την ανάλυση απαιτήσεων στη σύνταξη των προδιαγραφών και ο ρόλος του επικεφαλής μηχανικού (Software Architect)

Ζητούμενα από το κείμενο των τεχνικών προδιαγραφών

- Να προσδιορίσουν τον τρόπο με τον οποίο το λογισμικό θα ικανοποιήσει τις απαιτήσεις

Συγκεκριμένα

- Αρχιτεκτονικός σχεδιασμός και συστατικά του λογισμικού
- Σαφής περιγραφή της αλληλεπίδρασης του λογισμικού με το παραγωγικό του περιβάλλον (production environment)
- Λεπτομερής προδιαγραφή των λειτουργιών και των διαδικασιών που υποστηρίζονται από το λογισμικό
- Σύνδεση των προδιαγραφών με τις απαιτήσεις (πηγή/ προέλευση προδιαγραφής)
- Καθορισμός κριτηρίων αποδοχής / απόρριψης (τμημάτων ή όλου) του συστήματος

Έλεγχοι αποδοχής (acceptance tests)

- Έλεγχοι που καθορίζουν την αποδοχή ή την απόρριψη συστατικών ή λειτουργιών ή "οθονών" του λογισμικού
- Είναι καλή πρακτική να έχουν προβλεφθεί (μερικώς ή πλήρως) στις προδιαγραφές
- Ένας από τους κύριους στόχους ένταξης ελέγχων στον κύκλο του λογισμικού είναι η ελαχιστοποίηση των σφαλμάτων που θα προκύψουν κατά τους ελέγχους αποδοχής

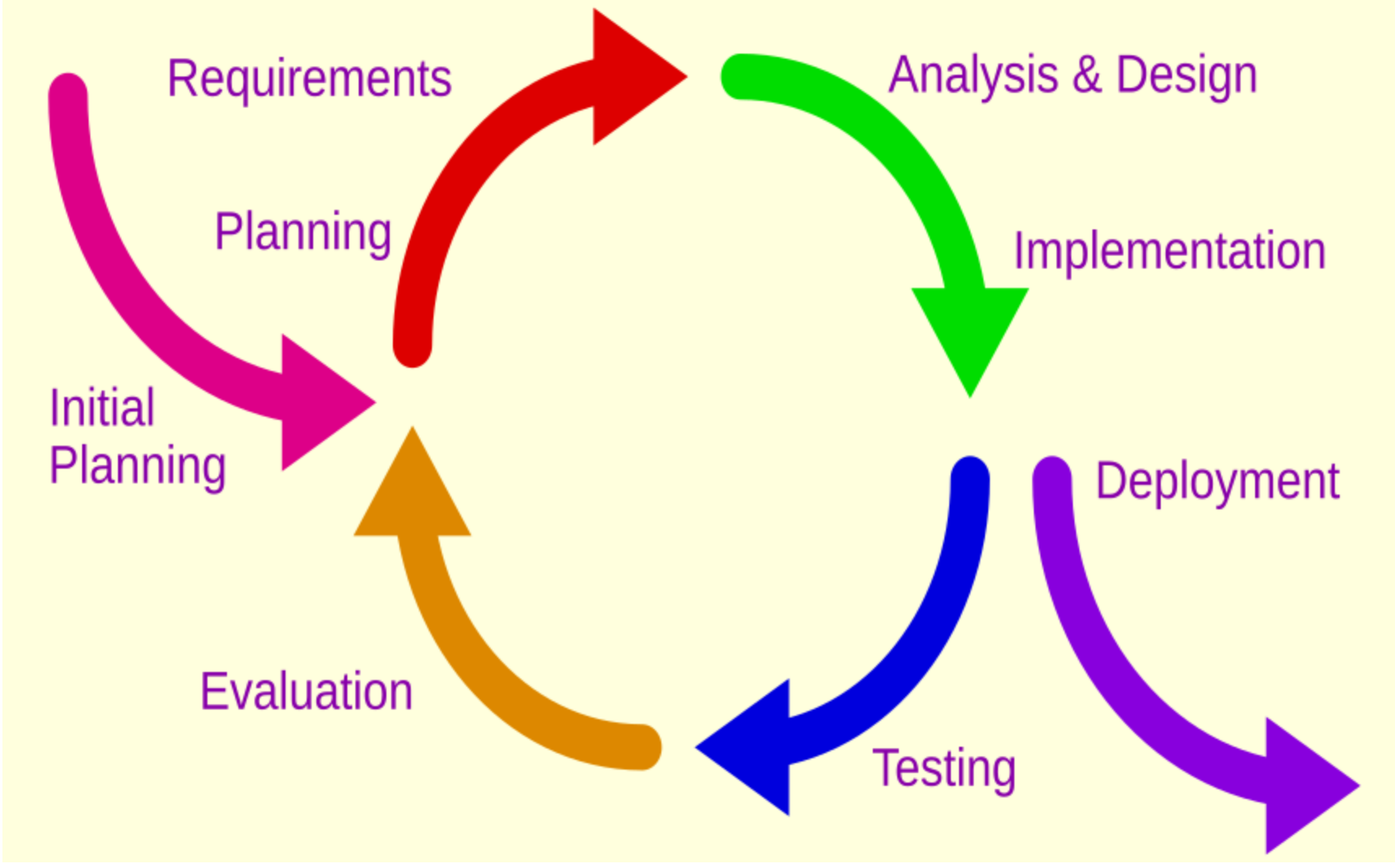
Σχεδιαστικές αποφάσεις

- Για να μετατραπούν οι υψηλού επιπέδου απαιτήσεις του λογισμικού
- Σε χαμηλού επιπέδου τεχνικές προδιαγραφές αυτού
- Θα πρέπει να παρθούν συγκεκριμένες σχεδιαστικές αποφάσεις

Δηλαδή

- Οι τεχνικές προδιαγραφές μοντελοποιούν τις απαιτήσεις εντός ενός οριοθετημένου σχεδιαστικού / αρχιτεκτονικού πλαισίου
- Δεν είναι πάντα εφικτό ο σχεδιασμός / αρχιτεκτονική να έχουν ολοκληρωθεί μέχρι τελευταίας λεπτομέρειας
- Από την άλλη δεν είναι αποδεκτό να προκύψει στην πορεία υλοποίησης μιας προδιαγραφής ένα ζήτημα που θα "ακυρώσει" το σχεδιασμό
- Το ότι θα τον αλλάξει είναι (σχεδόν) βέβαιο!

Θυμηθείτε


Requirements
Analysis & Design
Planning
Implementation
Initial
Planning
Deployment
Evaluation
Testing

Κοινός τόπος

- Η ανάλυση των απαιτήσεων πρέπει να καταλήξει σε ένα κοινά αποδεκτό ζητούμενο
- Κάτι που μπορεί να είναι δύσκολο στην πράξη για μη τεχνικούς λόγους (επικοινωνίας, οργάνωσης, διαφορετικής κουλτούρας, παγίωσης μη βέλτιστων πρακτικών, κτλ.)

Θυμηθείτε από την εισαγωγή

Στα μεγάλα έργα λογισμικού εμπλέκονται πολλοί συμμετέχοντες:

- Χρήστες
- Πελάτες
- Διοίκηση (ακόμα και Μέτοχοι ή Επενδυτές)
- Αναλυτές
- Προγραμματιστές
- Δοκιμαστές
- Σχεδιαστές διεπαφών
- Υπεύθυνοι έργου
- κτλ.

Ο ρόλος του Software Architect

Διαχείριση της πολυπλοκότητας

- 50% αρχιτεκτονική (ουσιώδης πολυπλοκότητα)
- 50% επικοινωνία (τεχνητή πολυπλοκότητα)
  - με τους stakeholders
  - με την επιχειρησιακή ομάδα έργου (Project Manager, Business Analyst, κ.ά)
  - με την ομάδα ανάπτυξης (engineers)

Κοινή γλώσσα

- Κάθε ένας από τους συμμετέχοντες έχει διαφορετικές εμπειρίες, βιώματα, αφετηρίες, στόχους και επιδιώξεις από την υλοποίηση του έργου
- Η ομάδα ανάπτυξης, η επιχειρησιακή ομάδα και οι stakeholders του έργου μάλλον "θα μιλάνε" σε διαφορετικές γλώσσες
- Ο Architect θα πρέπει να κάνει το διερμηνέα και να "μιλάει" τη γλώσσα και των τριών

Ιεράρχηση απαιτήσεων

- Δεν είναι όλες οι απαιτήσεις εξίσου σημαντικές
- Μια μικρή -θεωρητικά- απαίτηση μπορεί να οδηγήσει το έργο εκτός δρόμου/στόχου
  - Η ομάδα ανάπτυξης ή η επιχειρησιακή ομάδα μπορεί να μη δώσουν προσοχή ("αφού το ζητάει ο πελάτης")
  - Ο stakeholder ενδέχεται να μην το καταλάβει
- Ο Architect πρέπει να το διαγνώσει έγκαιρα και να προτείνει εναλλακτικές

Αντίστοιχα

- Μια "ακριβή" απαίτηση μπορεί να έχει πιο "φθηνή" εναλλακτική
- Μια "σύνθετη" απαίτηση μπορεί να έχει πιο "απλή" εναλλακτική
- κ.ο.κ

Προσοχή

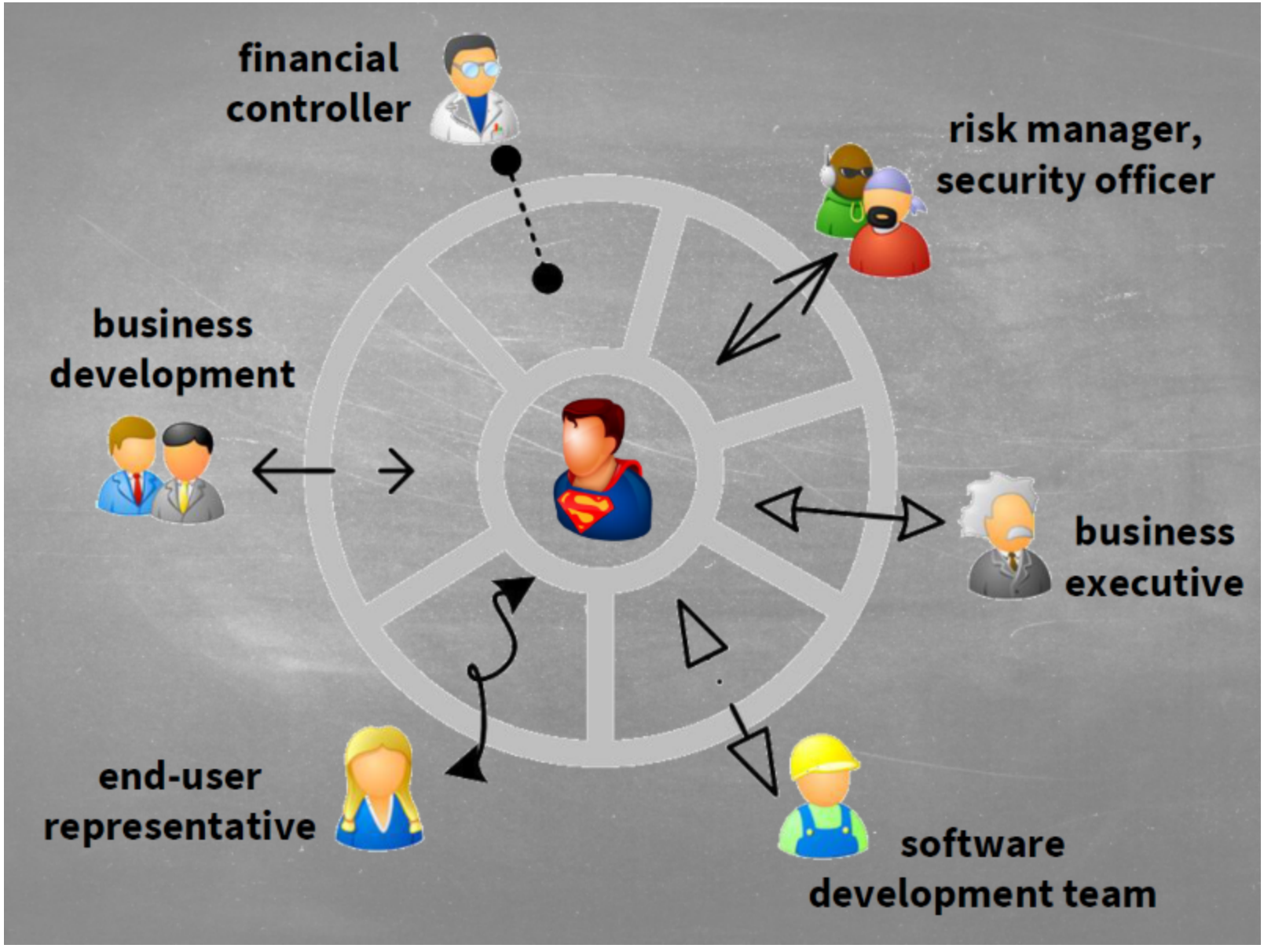
- Ο ασφαλέστερος τρόπος να αποτύχει ένα έργο είναι να επιτραπεί στις "λάθος" απαιτήσεις να γίνουν προδιαγραφές!

Συμπέρασμα

- Ο Architect θα πρέπει να έχει απευθείας επαφή με τους stakeholders
  - Αναλύοντας και διαμορφώνοντας μαζί με την επιχειρησιακή ομάδα του έργου τις απαιτήσεις
- Ενώ παράλληλα σχεδιάζει το σύστημα
  - Για να είναι σε θέση να φιλτράρει τις "λάθος" απαιτήσεις
  - Και να μεταφράσει τις υπόλοιπες σε προδιαγραφές για την ομάδα ανάπτυξης

> The belief that complex systems require armies of programmers and designers is wrong. A system that is not understood in its entirety, or at least to a significant degree of detail by a single individual, should probably not be built.
>
> N. Wirth

Superman



By J.Schulenklopper, E. Rommes

Για να γίνεις Superman

Ενδεικτικά

- Ενσυναίσθηση (empathy)
  - Για τις έγνοιες (concerns) όλων των εμπλεκόμενων
- Τεχνική αρτιότητα
  - Επιλογή τεχνολογίας που να είναι συμβατή με τις ανάγκες του έργου και τις ικανότητες της ομάδας
- Επικοινωνιακή ικανότητα
  - Συνεχής επικοινωνία με όλους τους εμπλεκόμενους για την εύρεση των βέλτιστων συμβιβασμών (trade-offs)
- Εμπειρία
  - Για να μπορούν να γίνουν καλά όλα τα παραπάνω

3. Οπτικοποίηση και μοντελοποίηση απαιτήσεων / προδιαγραφών & σχεδιασμού

Μοντελοποίηση και οπτικοποίηση απαιτήσεων/προδιαγραφών

Επικοινωνία των ζητούμενων

- Διαγράμματα ροής (flow charts)
- Διαγράμματα ροής δεδομένων (data flow diagrams)
- Πίνακες απόφασης (decision tables)
- Δέντρο απόφασης (decision trees)
- Mind Maps

Μοντελοποίηση σχεδιασμού

Επικοινωνία του συστήματος και των συστατικών του

- **Αντικειμενοστραφές μοντέλο**
- Ψευδοκώδικας
- Διαγράμματα οντοτήτων-συσχετίσεων (Entity-Relationship diagrams)
- Οντολογίες (ontologies)
- **Πρότυπα σχεδίασης (design patterns)**

UML

Ενοποιημένη γλώσσα μοντελοποίησης

- Μοντελοποίηση απαιτήσεων και προδιαγραφών
- Μοντελοποίηση σχεδιασμού

Γιατί αυτός ο διαχωρισμός;

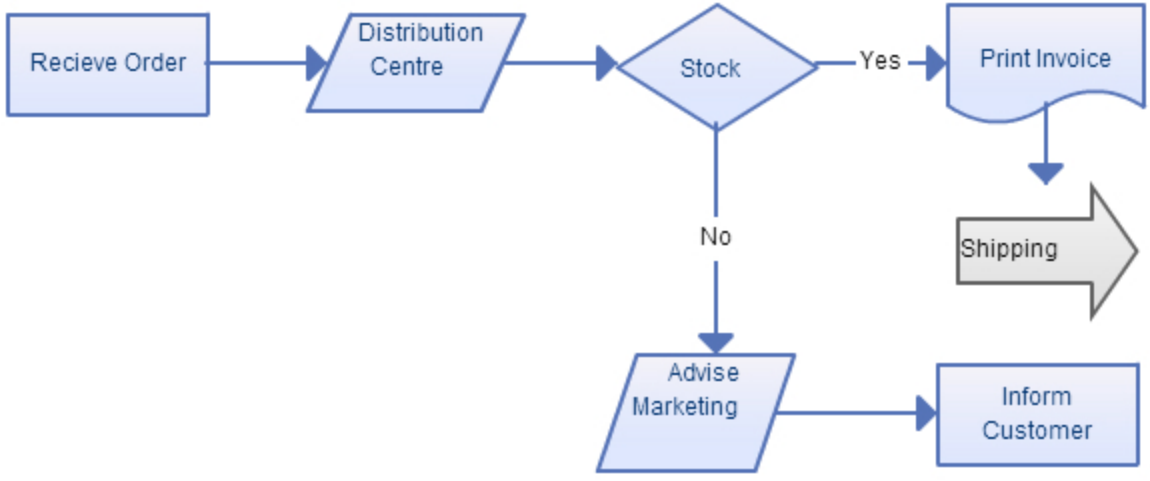
> It is almost always incorrect to begin the decomposition of a system into modules on the basis of a flowchart. We propose instead that one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others. Since, in most cases, design decisions transcend time of execution, modules will not correspond to steps in the processing.
>
> David Parnas

Η έμφασή μας στο μάθημα

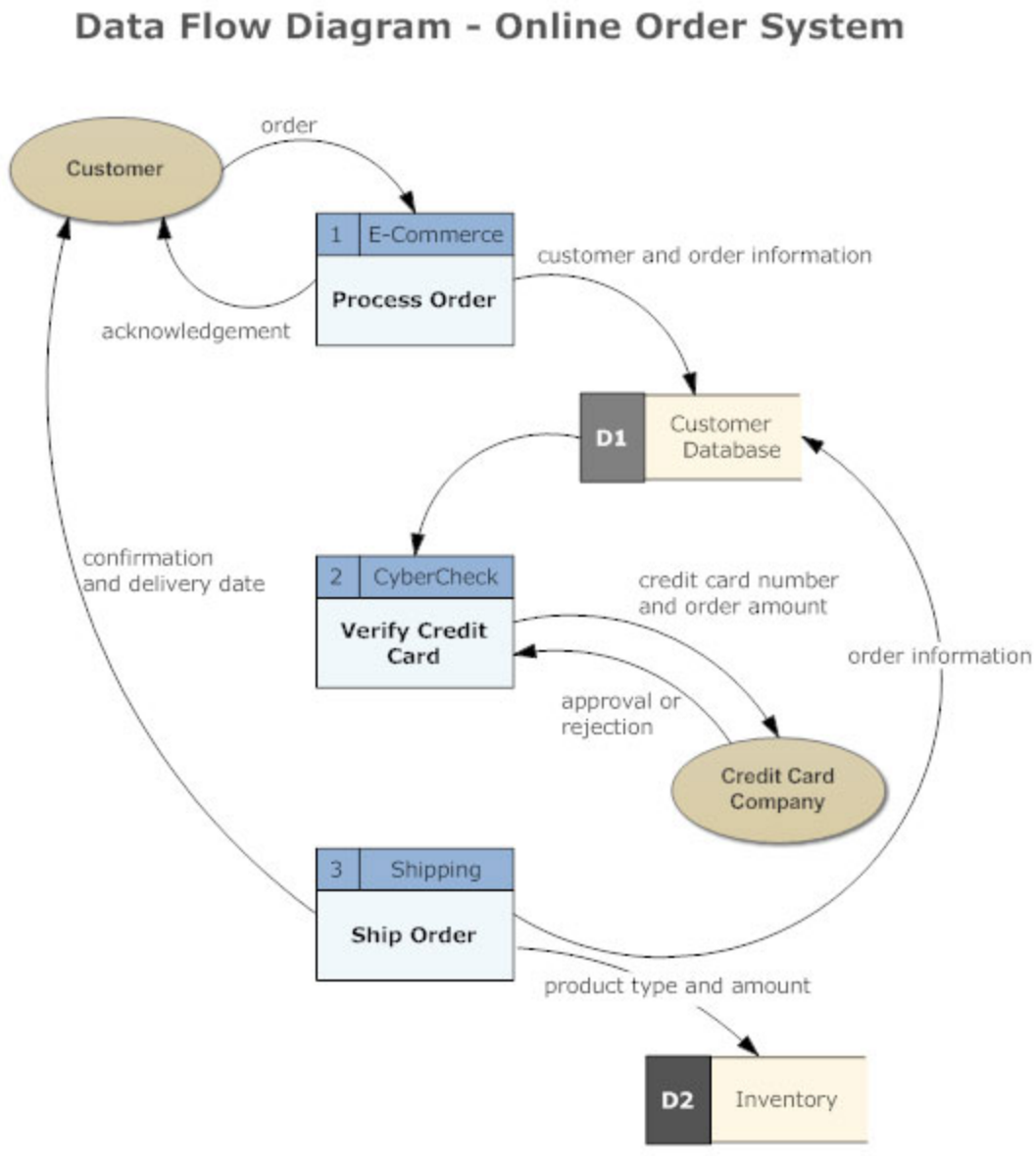
Σε επόμενες διαλέξεις

- Αντικειμενοστραφές μοντέλο
- UML
- Design Patterns

Διάγραμμα ροής (flow chart)

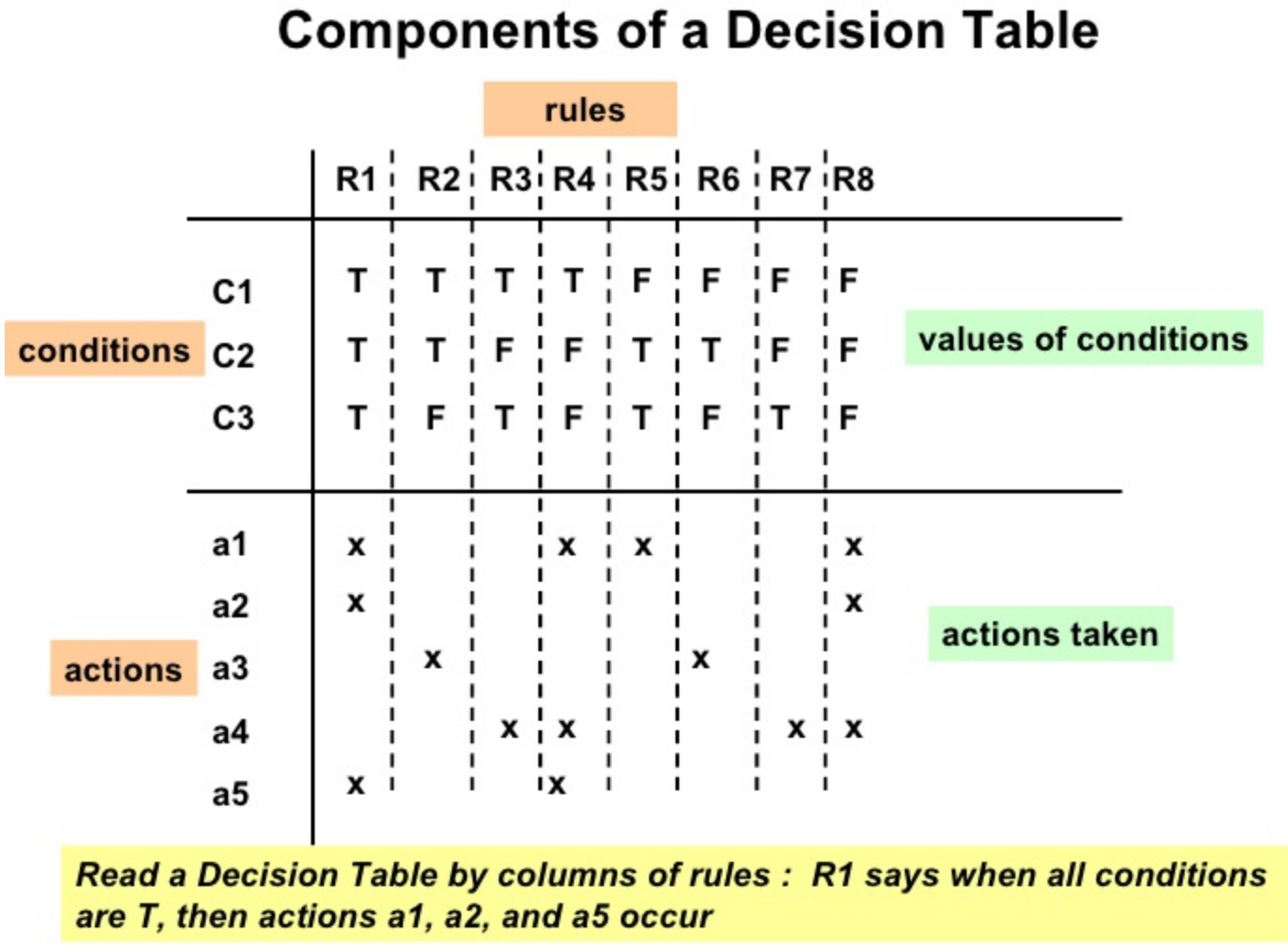


Διάγραμμα ροής δεδομένων (data flow diagram)



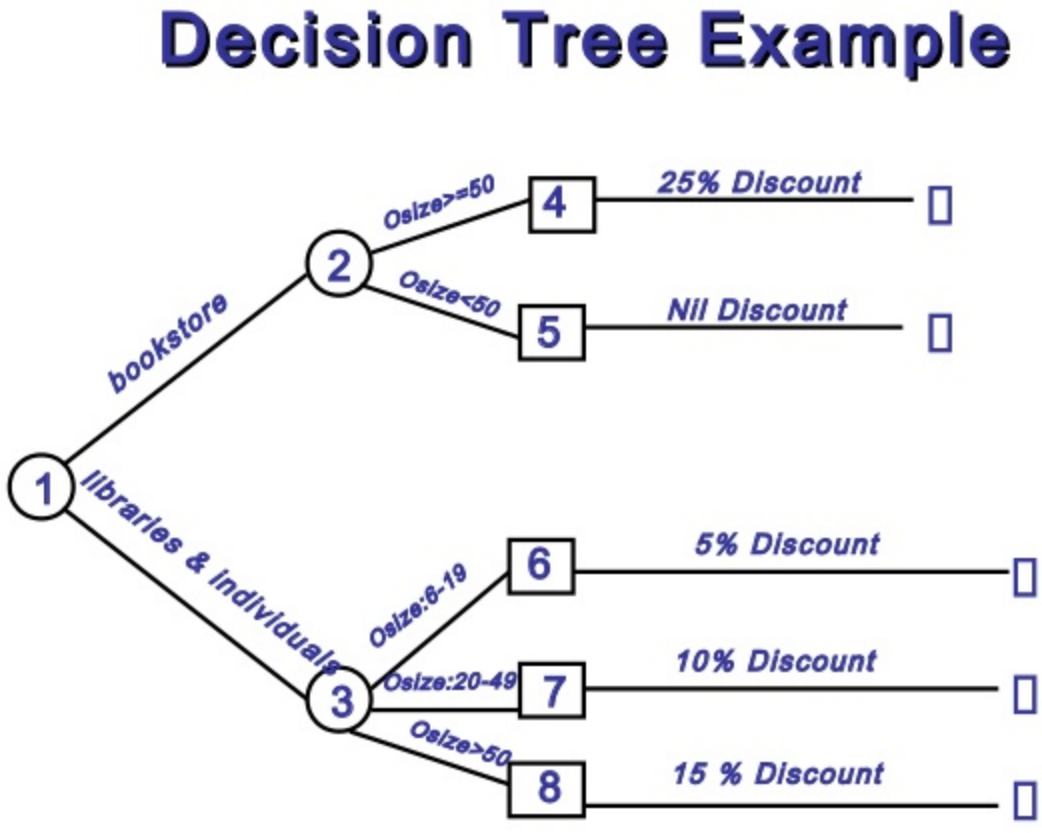
Πίνακας αποφασης (decision table)

Με εφαρμογή στον έλεγχο λογισμικού (testing)

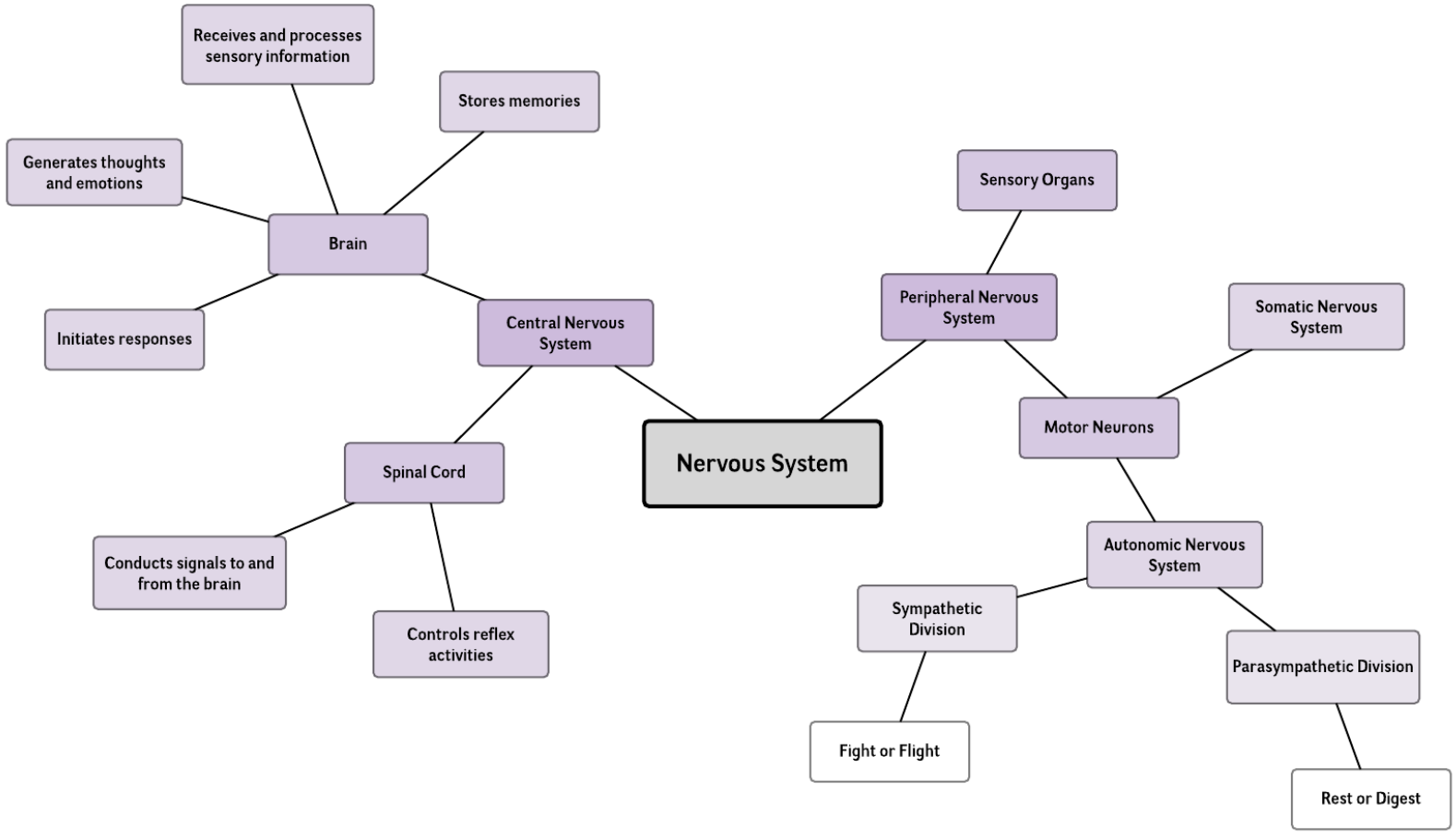


Δέντρο απόφασης (decision tree)

Με εφαρμογές στην επιχειρησιακή έρευνα και στη μηχανική μάθηση



Mind Map

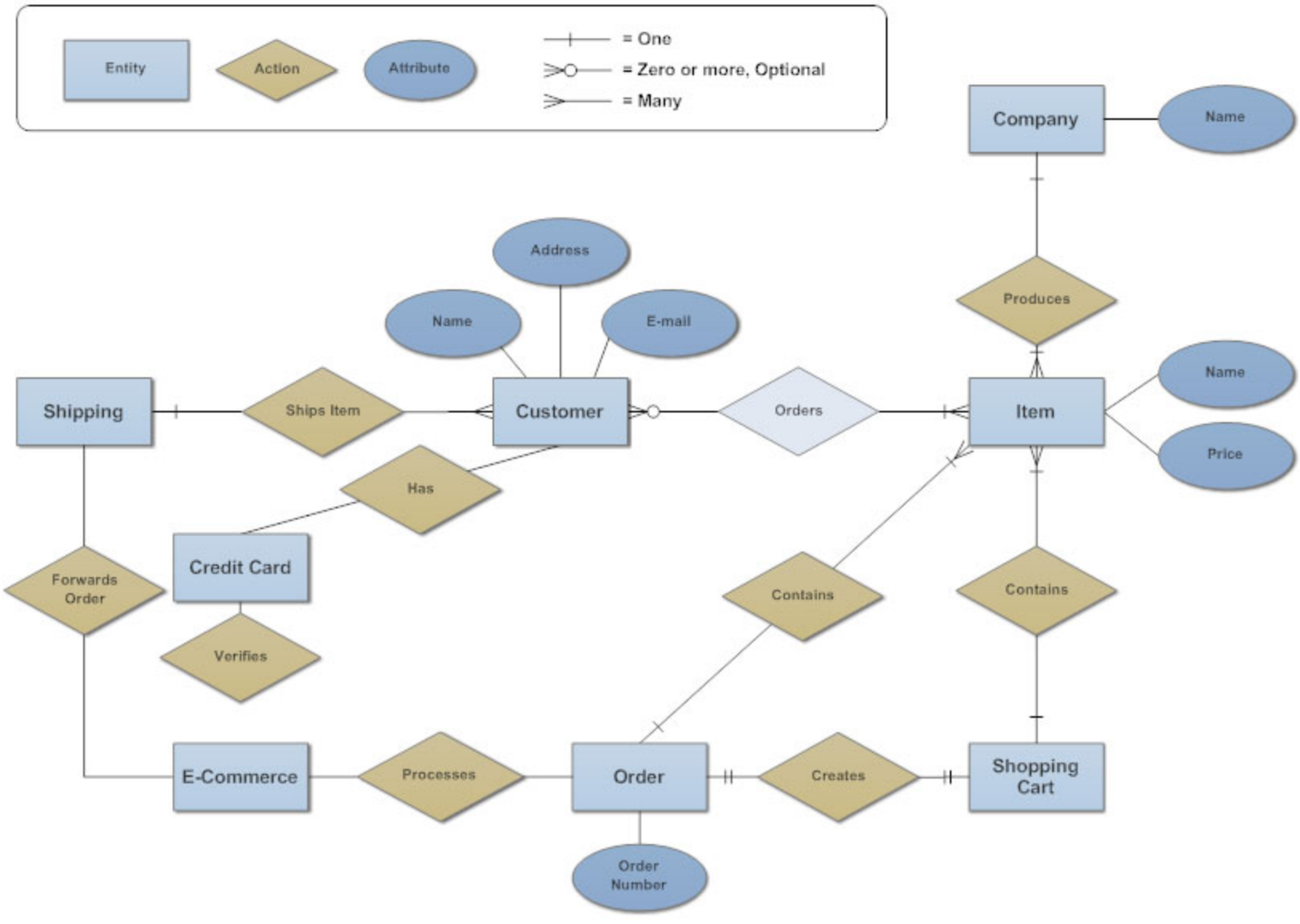

Nervous System
Brain
Receives and processes sensory information
Stores memories
Generates thoughts and emotions
Initiates responses
Central Nervous System
Spinal Cord
Conducts signals to and from the brain
Controls reflex activities
Peripheral Nervous System
Sensory Organs
Somatic Nervous System
Motor Neurons
Autonomic Nervous System
Sympathetic Division
Fight or Flight
Parasympathetic Division
Rest or Digest

Ψευδοκώδικας

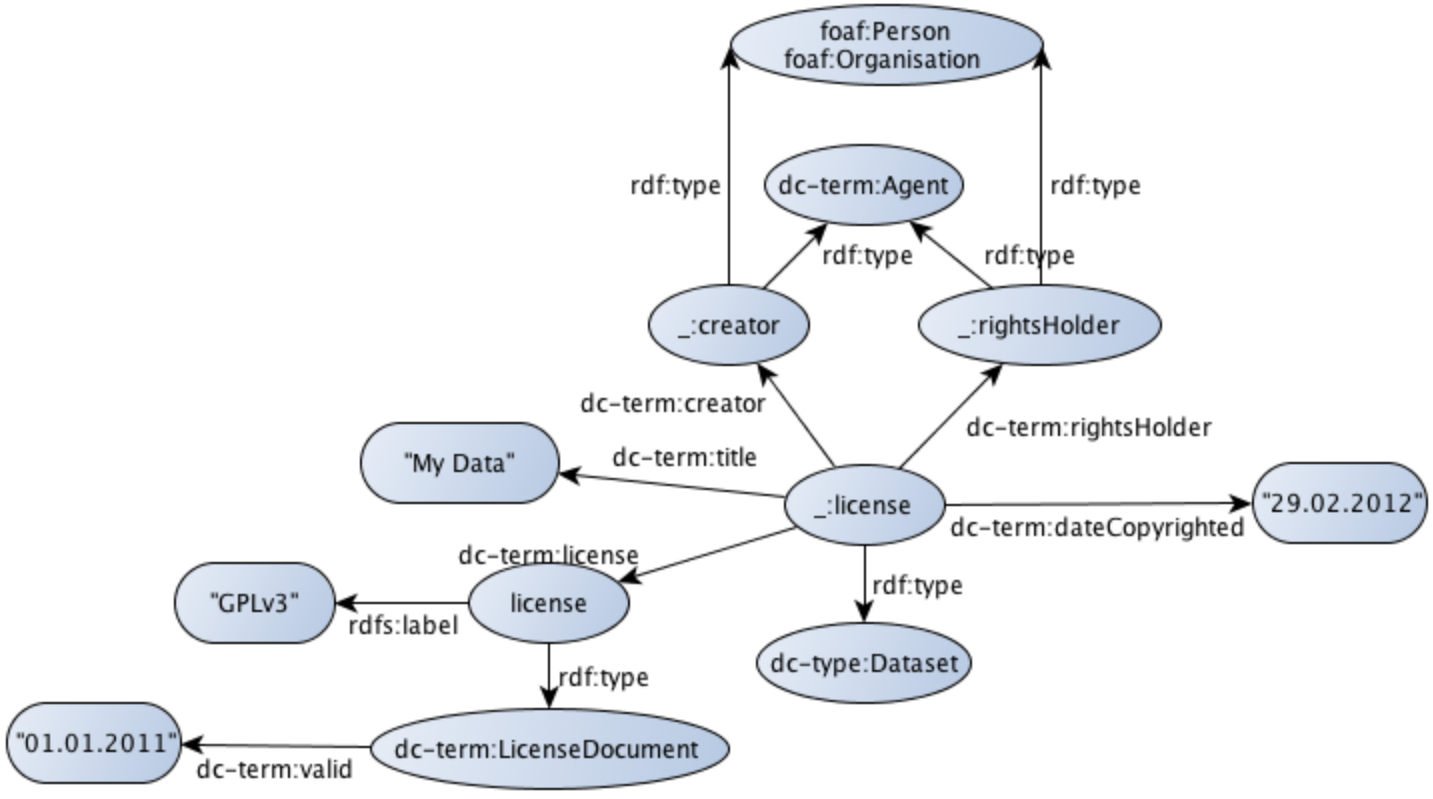
Algorithm 1: Euclid's algorithm for finding the greatest common divisor of two nonnegative integers

```
1 function Euclid (a, b);
  Input  : Two nonnegative integers a and b
  Output: gcd(a, b)
2 if b = 0 then
3 |   return a;
4 else
5 |   return Euclid(b, a mod b);
6 end
```

Entity Relationship Diagram - Internet Sales Model



Οντολογία (ontology)



4. Σχεδιαστικές αρχές λογισμικού

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)
  Map<String, Object> toMap()
  void fromMap(Map<String, Object> map)
}

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 Datastore {
  boolean exists(String id)
  void load(String id, Function<Item> callback)
  void save(Item item, Function<Item> callback)
}
```

Refactoring

- Η διαδικασία επαναπροσαρμογής του κώδικα ως τμήμα κάποιας βελτιωτικής απόφασης
- Αυτοματοποιείται από πολλά IDEs
- Θα τη δούμε λεπτομερώς 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

> 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

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)

Ενθυλάκωση (encapsulation)

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

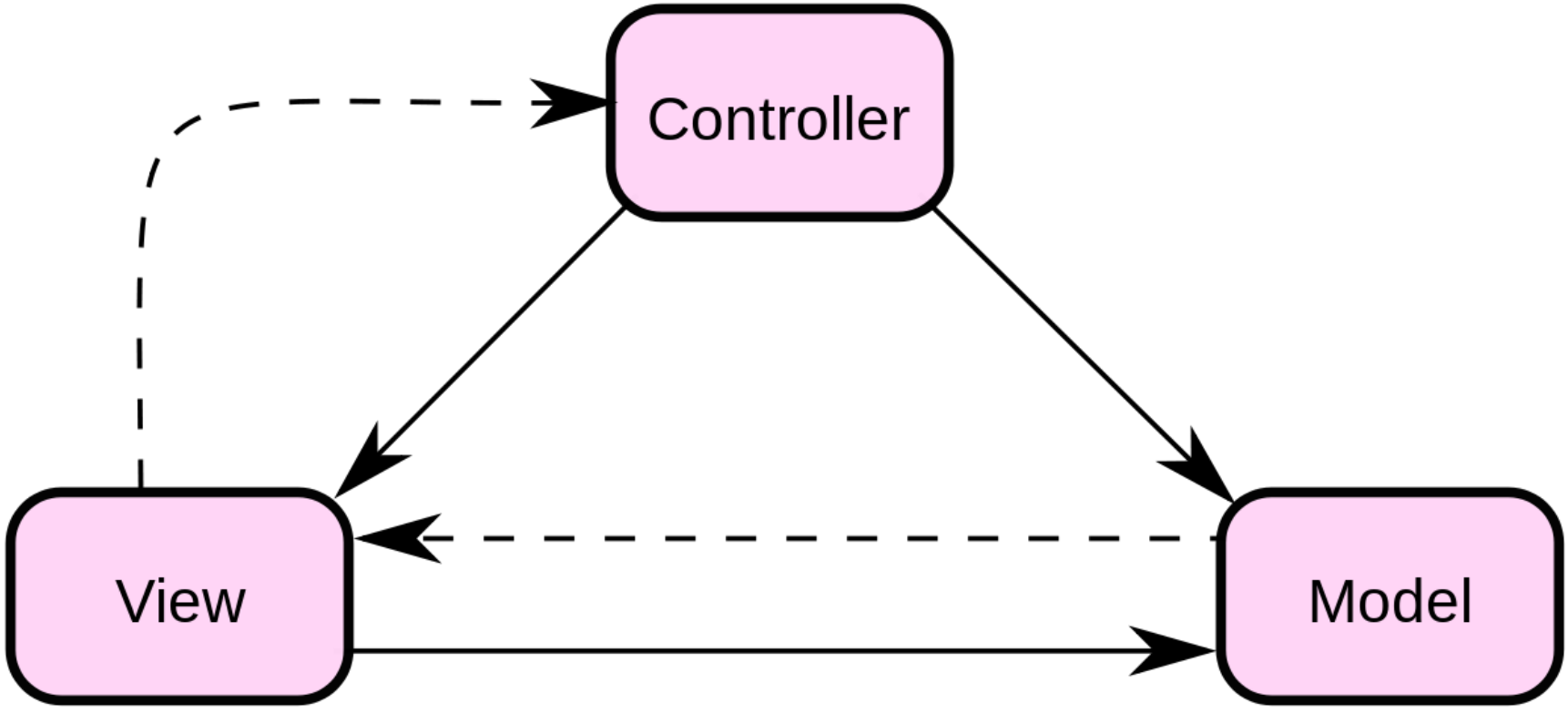
Παράδειγμα

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

Διαχωρισμός ενδιαφερόντων (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 επανέλθουμε

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

```
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
```

Μήπως ο ίδιος ο κώδικας είναι το design;

> TEX would have been a complete failure if I had merely specified it and not participated fully in its initial implementation. The process of implementation constantly led me to unanticipated questions and to new insights about how the original specifications could be improved.
>
> Donald Knuth

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

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

Ανοικτό/κλειστό

- 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

- 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

- 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

- 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 Function<Item> updateIndex = ...
  //private MySQLDatastore store //bad
  private Datastore store        //good
  DataAccessLayer() {
    store = new MySQLDatastore()    //bad
    store = Config.get('Datastore') //good
  }
  void save(Item item) {
    store.save(Item item, updateIndex)
  }
}
```

Dependency Inversion (Injection)

5. Η έννοια του τεχνικού χρέους (technical/design/code debt)

Ποιοτικά ζητούμενα σχεδιασμού

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

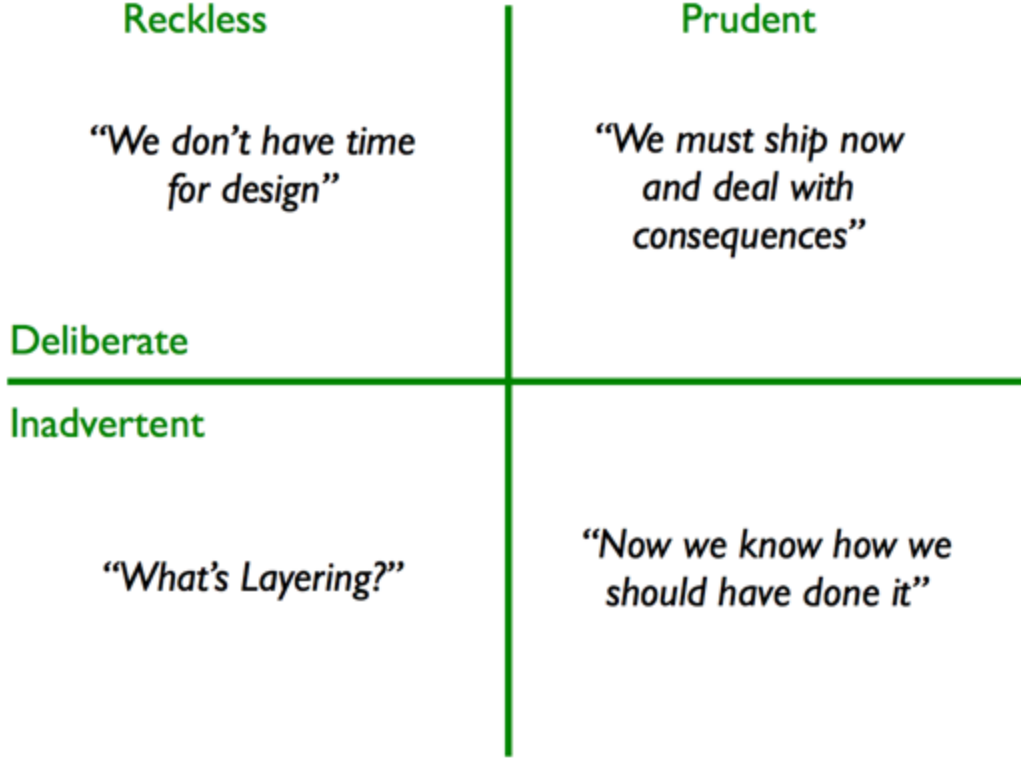
Σχεδιασμός = Συμβιβασμός

- Συνήθως δεν είναι εφικτό να γίνουν όλα καλά με την πρώτη
- Επιλογή των επιθυμητών trade-offs

Τεχνικό χρέος

- Το κόστος της πρόσθετης δουλειάς που θα απαιτηθεί από την επιλογή μιας εύκολης και γρήγορης υλοποίησης αντί για την εφαρμογή της συνολικά καλύτερης λύσης.

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