Αριστοτέλειο
Πανεπιστήμιο
Θεσσαλονίκης

ΑΝΤΙΚΕΙΜΕΝΟΣΤΡΕΦΗΣ ΑΝΑΛΥΣΗ

Η μεθοδολογία ICONIX

Ιωάννης Σταμέλος
Βάιος Κολοφωτιάς
Πληροφορική

Θεσσαλονίκη, Σεπτέμβριος 2013

Άδειες Χρήσης

Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons. Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.

Χρηματοδότηση

Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα. Το έργο «**Ανοικτά Ακαδημαϊκά Μαθήματα στο Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης**» έχει χρηματοδοτήσει μόνο τη αναδιαμόρφωση του εκπαιδευτικού υλικού.

ανοικτάμαθήματα
opencourses

Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο

ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ & ΘΡΗΣΚΕΥΜΑΤΩΝ, ΠΟΛΙΤΙΣΜΟΥ & ΑΘΛΗΤΙΣΜΟΥ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ

ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ

Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Περιεχόμενα

1. Περιεχόμενο Μαθήματος

Εβδομάδα	**Περιεχόμενο**
1η	Εισαγωγή στην Αντικειμενοστρεφή Ανάλυση/UML
2η	Rational Unified Process
3η	Περιπτώσεις Χρήσης
4η	Διαγράμματα Κλάσεων
5η	Διαγράμματα Συνεργασίας
6η	Διαγράμματα Ακολουθίας
7η	Πρότυπα Σχεδίασης
8η	**Διεργασία ICONIX**
9η	Επιχειρηματική Μοντελοποίηση
10η	Υλοποίηση Σχεδίασης με Java
11η	Μετρικές Αντικειμενοστραφούς Σχεδίασης
12η	Επισκόπηση

1.1 Περιεχόμενα ενότητας

Στην ενότητα αυτή θα εξετάσουμε την μεθοδολογία ICONIX η οποία είναι απλούστερη και συντομότερη από την RUP, που εξετάσαμε νωρίτερα. Στη μεθοδολογία αυτή χρησιμοποιούμε μόνο 4 UML διαγράμματα. Ακολουθούμε τα βήματα του σχεδιασμού και της ανάλυσης απαιτήσεων και στη συνέχεια κάνουμε επισκόπηση του σχεδιασμού με πλήθος ελέγχων για την ποιότητα, το συντακτικό την σημασιολογία κ.α

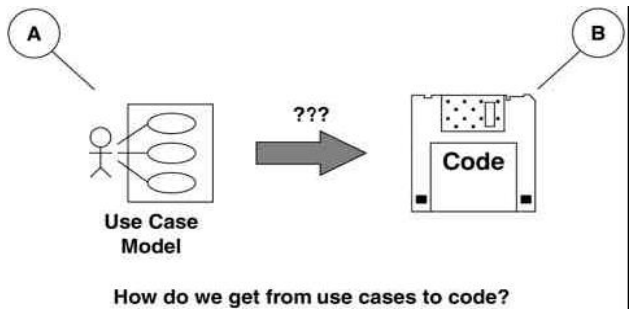
1.2 Η μεθοδολογία - Χαρακτηριστικά

- Η ICONIX:
  - είναι απλούστερη και συντομότερη από την Rational Unified Process (RUP),
  - υιοθετεί την UML ως γλώσσα έκφρασης των απαιτήσεων και των προδιαγραφών του υπό σχεδίαση λογισμικού,
  - είναι «καθοδηγούμενη από τις περιπτώσεις χρήσης»,
  - αποφεύγει την πληθώρα μοντέλων χωρίς να παραλείπει τις διαδικασίες ανάλυσης και σχεδιασμού, παρέχει σε κάθε βήμα τη δυνατότητα ανίχνευσης του βαθμού υλοποίησης των απαιτήσεων – δεν επιτρέπει σε κανένα σημείο την απομάκρυνση από τις ανάγκες του χρήστη,
  - Είναι επαναληπτική και αυξητική – το στατικό μοντέλο εκλεπτύνεται καθώς αναλύεται το δυναμικό μοντέλο – χωρίς να επιφέρει μεγάλη διαχειριστική επιβάρυνση.

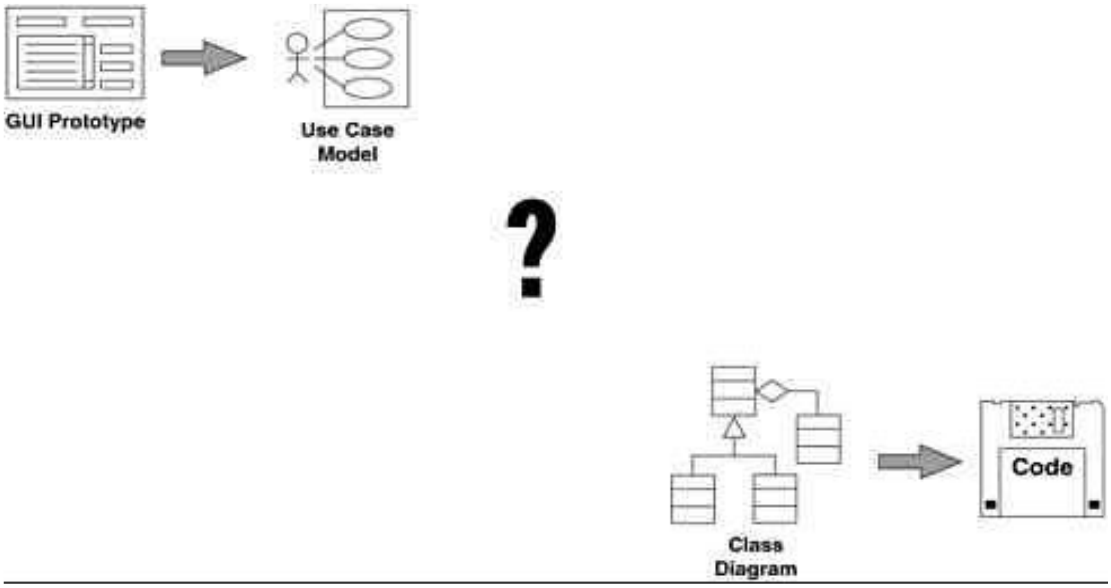
2. Διαγράμματα ICONIX

Για να εφαρμόσουμε την ICONIX θα χρησιμοποιήσουμε μόνο τέσσερα UML διαγράμματα:

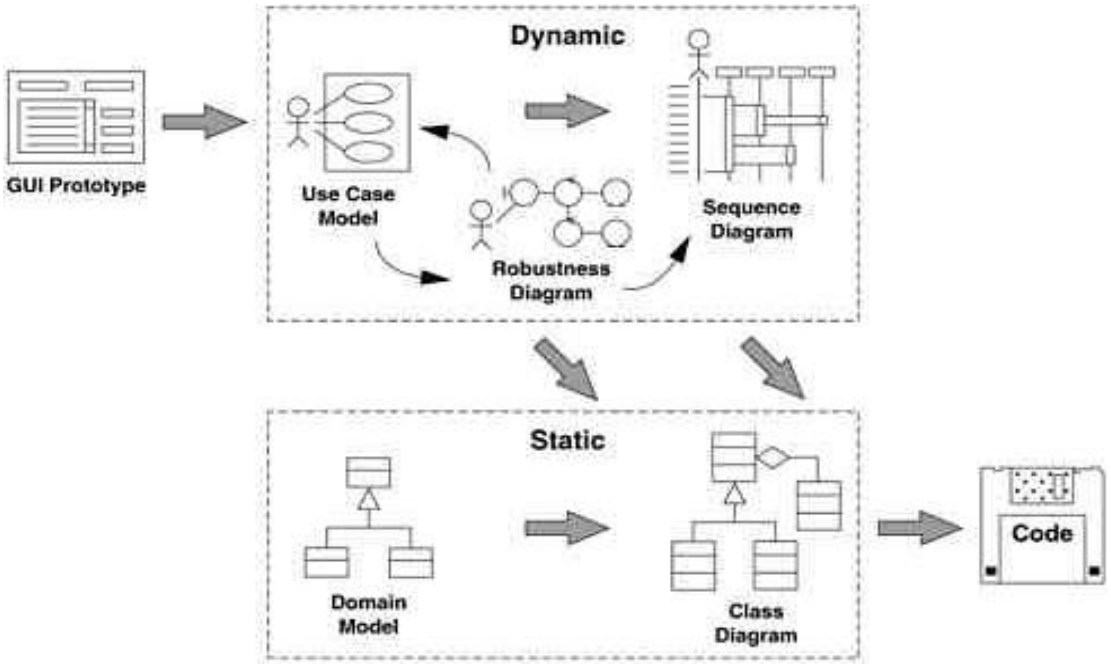
- **Διάγραμμα περιπτώσεων χρήσης** για να αναπαραστήσουμε τα σενάρια χρήσης και τους χειριστές του συστήματος
- **Διάγραμμα κλάσεων** για να αναπαραστήσουμε το πεδίο εφαρμογής του συστήματος αλλά και τη λεπτομερή στατική δομή του συστήματος
- **Διάγραμμα συνεργασίας** για να αναπαραστήσουμε τον τρόπο με τον οποίο η στατική δομή υλοποιεί τα σενάρια χρήσης του συστήματος
- **Διάγραμμα ακολουθίας** για να συσχετίσουμε λεπτομερώς τη δυναμική συμπεριφορά με τη στατική δομή του συστήματος



Εικόνα 1 : Use case to model



Εικόνα 2 : Αρχή

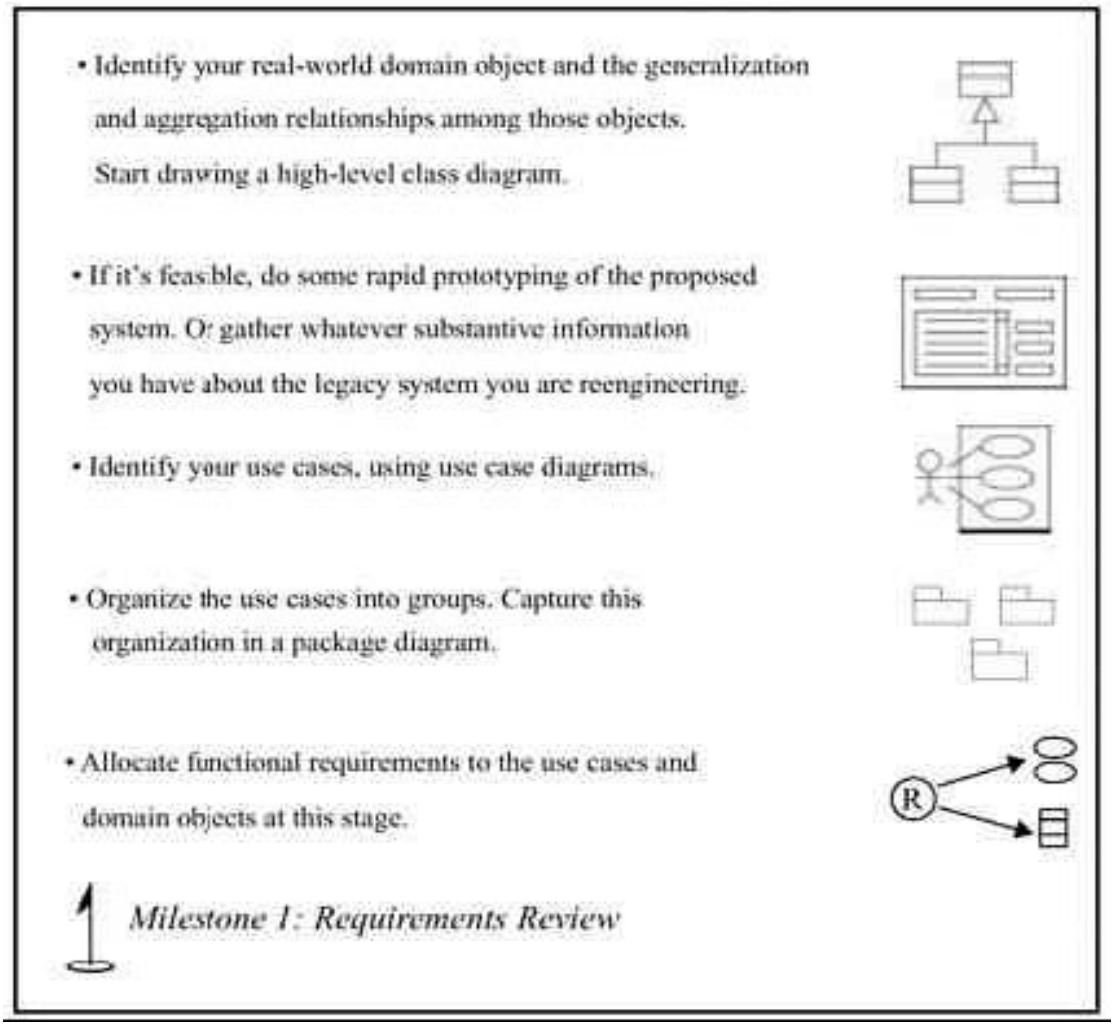


Εικόνα 3 : Δομή

2.1 Βήματα

- Βήμα 1. Ανάλυση απαιτήσεων
  - Βήμα 1.1 Αναπαράσταση πεδίου εφαρμογής
  - Βήμα 1.2. Σχεδίαση περιπτώσεων χρήσης
    - Σημείο ελέγχου 1. Επισκόπηση απαιτήσεων
    - Βήμα 2. Ανάλυση ευρωστίας
    - Σημείο ελέγχου 2. Προκαταρκτική επισκόπηση σχεδιασμού
  - Βήμα 3. Λεπτομερής σχεδιασμός
    - Σημείο ελέγχου 3. Κρίσιμη επισκόπηση σχεδιασμού

2.1.1 Ανάλυση Απαιτήσεων



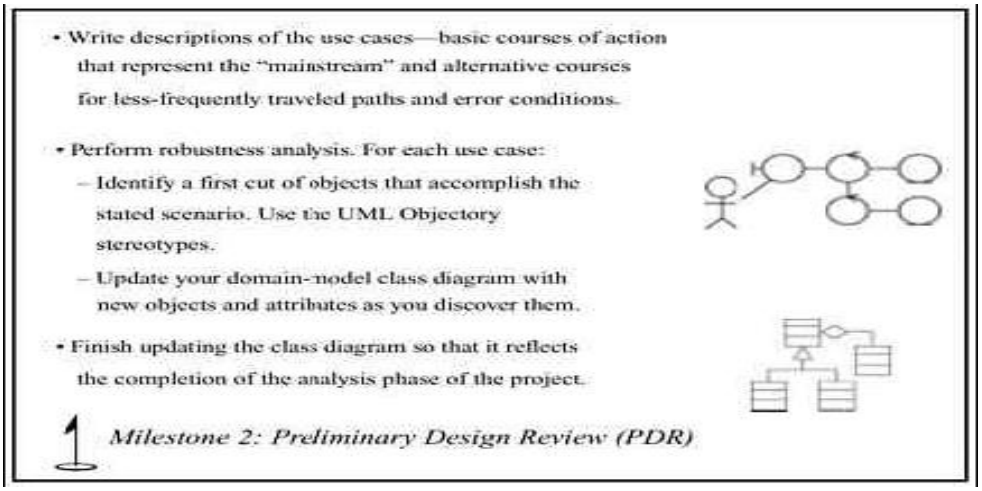
Εικόνα 4 : Requirements Review

2.1.2 Αναθεώρηση Απαιτήσεων(Requirements Review)

Μία ομάδα τεχνολόγων διαβάζει,απαιτήσεις,περιπτώσεις χρήσης κλπ και επισημαίνει λάθη, παραλείψεις, ασάφειες, επαναλήψεις, ασυνέπειες κλπ

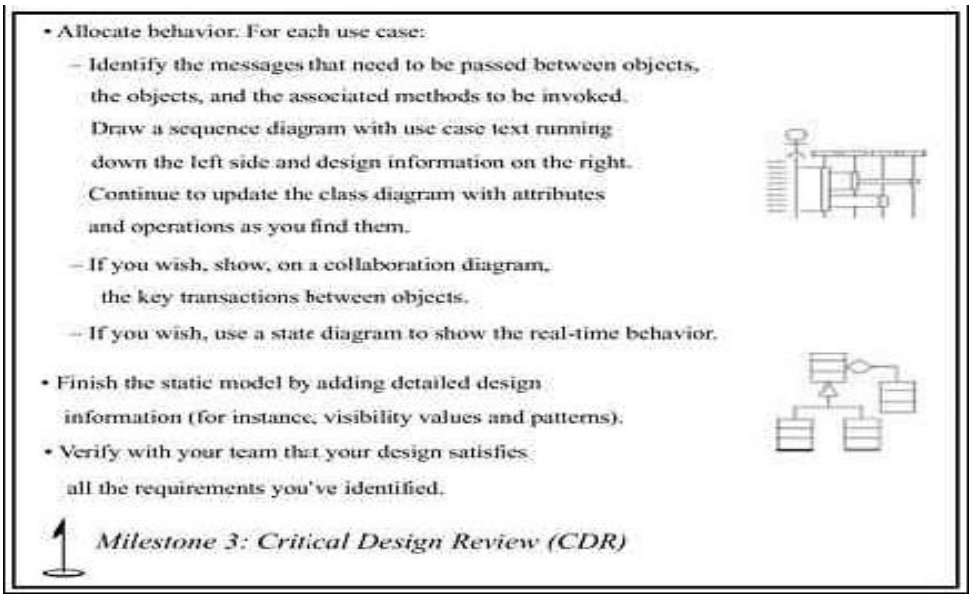
Θεωρείται ο καλύτερος τρόπος έγκαιρου εντοπισμού προβλημάτων σε ένα έργο λογισμικού

2.1.2.1 Ανάλυση Ευρωστίας



Εικόνα 5 Ανάλυση Ευρωστίας

2.1.2.2 Λεπτομερής σχεδιασμός



Εικόνα 6 Λεπτομερής Σχεδιασμός

3. Εφαρμογή

Φροντίζουμε να περιγράψουμε τη στατική δομή και τη δυναμική συμπεριφορά του συστήματος

- Στατική δομή: αναλυτικό διάγραμμα κλάσεων
- Συμπεριφορά: διαγράμματα ακολουθίας

Ξεκινούμε από ένα αρχικό στατικό μοντέλο, το οποίο συνεχώς εμπλουτίζουμε

- Αρχικά είναι το μοντέλο του πεδίου εφαρμογής ενώ στην πορεία γίνεται το μοντέλο της δομής του συστήματος

Περιγράφουμε το σύστημα υπό μορφή σεναρίων λειτουργίας και οθονών

- Για κάθε σενάριο, περιγράφουμε ένα σύνολο περιπτώσεων χρήσης
- Για κάθε περίπτωση, σχεδιάζουμε ένα διάγραμμα συνεργασίας
- Για κάθε διάγραμμα, σχεδιάζουμε ένα διάγραμμα ακολουθίας
- Σε κάθε βήμα εμπλουτίζουμε την περιγραφή των περιπτώσεων χρήσης και το στατικό μοντέλο

Αναλύουμε και σχεδιάζουμε το σύστημα μέσα στα πλαίσια που ορίζει το μοντέλο του πεδίου εφαρμογής

3.1 Κρίσιμη Επισκόπηση Σχεδιασμού (Critical Design Review)

Πρόκειται για μία σειρά ελέγχων που διασφαλίζουν την ποιότητα της ανάλυσης και σχεδίασης (εντοπίζουν τυχόν λάθη με συστηματικό τρόπο)

Πρέπει να επιβεβαιωθούν τα εξής:

3.1.1 Έλεγχοι CDR:

Για κάθε περίπτωση χρήσης, η ροή μηνυμάτων στο διάγραμμα ακολουθίας πρέπει να ταιριάζει με την ακολουθία ενεργειών στην περιγραφή της περίπτωσης χρήσης, για τη βασική και όλες τις εναλλακτικές ακολουθίες ενεργειών και αποκρίσεων

Ελέγχεται η ορθότητα της ροής ελέγχου σε κάθε διάγραμμα ακολουθίας, ελέγχοντας την ορθότητα της κατεύθυνσης των μηνυμάτων και τη διασφάλιση της συνεχούς λειτουργίας του συστήματος.

- Κάθε φορά πρέπει να είναι εμφανές ποιο αντικείμενο στέλνει και ποιο λαμβάνει κάθε μήνυμα, ενώ δεν πρέπει να υπάρχει μεταφορά ελέγχου εκτέλεσης ανάμεσα σε αντικείμενα χωρίς την ανταλλαγή αντίστοιχου μηνύματος.

Έλεγχος της καλής κατανομής μεθόδων σε κλάσεις, καθώς αυτή επηρεάζει την ποιότητα των κλάσεων. **Χρησιμοποιούμε τα ακόλουθα κριτήρια**:

- **Επαναχρησιμοποιησιμότητα** (reusability): όσο ποιο γενική είναι μια κλάση, τόσο πιο εύκολα μπορούμε να την επαναχρησιμοποιήσουμε.
  - Πριν ενσωματώσουμε μια μέθοδο σε μια συγκεκριμένη κλάση, πρέπει να αναρωτηθούμε κατά πόσο αυτή θα επηρεάσει την δυνατότητα επαναχρησιμοποίησής της
- **Εφαρμοσιμότητα** (applicability): πρέπει κάθε φορά να εξετάζουμε κατά πόσο μια μέθοδος «ταιριάζει» με την κλάση στην οποία σκεφτόμαστε να την κατανείμουμε (cohesion – συνεκτικότητα της κλάσης).
- Για παράδειγμα, η μέθοδος θα πρέπει να χρησιμοποιεί δεδομένα της κλάσης και να υλοποιεί λειτουργίες που ταιριάζουν με αυτές που υλοποιούν οι υπόλοιπες μέθοδοι της κλάσης
- Πολυπλοκότητα (complexity): εξετάζουμε το βαθμό δυσκολίας υλοποίησης μιας μεθόδου μέσα στη μια ή την άλλη κλάση, προσπαθώντας να μειώσουμε την πολυπλοκότητα της υλοποίησης
- Εσωτερική γνώση υλοποίησης της κλάσης: εξετάζουμε εάν η υλοποίηση μιας μεθόδου μέσα σε μια κλάση απαιτεί τη γνώση εσωτερικών λεπτομερειών υλοποίησης της κλάσης

3.1.2 Έλεγχος της ποιότητας των κλάσεων

Ελέγχεται ο βαθμός στον οποίο η σχεδίαση των κλάσεων ικανοποιεί τα ακόλουθα κριτήρια:

- Σύζευξη (coupling)
- Συνοχή (cohesion)
- Επάρκεια (sufficiency)
- Πληρότητα (completeness)
- Πρωτογένεια (primitiveness)

3.1.3 Κριτήρια

- **Σύζευξη** (coupling): στοχεύουμε σε χαλαρή (μειωμένη) σύζευξη, ώστε να μεγιστοποιήσουμε τη συναρμολογησιμότητα (modularity) του συστήματος. Με άλλα λόγια, επιδιώκουμε κλάσεις που είναι όσο πιο ανεξάρτητες γίνεται
- **Συνοχή** (cohesion): στοχεύουμε σε υψηλό βαθμό λειτουργικής συνοχής ανάμεσα στα πεδία και τις μεθόδους μιας κλάσης. Με άλλα λόγια, αποφεύγουμε να σχεδιάσουμε κλάσεις που έχουν «διχασμένη προσωπικότητα»
- **Επάρκεια** (sufficiency): στοχεύουμε σε υψηλό βαθμό επάρκειας κάθε κλάσης, δηλαδή αυτή πρέπει να ενθυλακώνει αρκετά αντικείμενα του πεδίου εφαρμογής, ώστε να αποτελεί ένα αυτόνομο και λογικά πλήρες τμήμα του συστήματος, με το οποίο άλλα τμήματα μπορούν να επικοινωνούν
    - Ελέγχουμε κλάσεις με ένα μόνο ή και κανένα αντικείμενο

- **Πληρότητα** (completeness): στοχεύουμε σε υψηλό βαθμό πληρότητας, ώστε η κάθε κλάση να είναι επαναχρησιμοποιήσιμη σε παρόμοια πεδία εφαρμογής.
    - Αυτό σημαίνει ότι στη διεπαφή της κλάσης έχουμε φροντίσει να αναπαραστήσουμε όλους τους τρόπους με τους οποίους μπορεί να χρησιμοποιηθεί
    - Αποφεύγονται σταθερές που σε άλλο πεδίο προβλήματος μπορεί να έχουν διαφορετική τιμή (π.χ. Αριθμός τροχών οχήματος)
- **Πρωτογένεια** (primitiveness): ο στόχος είναι να σχεδιάσουμε ορισμένες βασικές μεθόδους, οι οποίες θα έχουν πρόσβαση σε έννοιες και αφαιρέσεις των μοντέλων του συστήματος, και τις οποίες στη συνέχεια θα χρησιμοποιήσουμε ως δομικά υλικά σε περισσότερο πολύπλοκες μεθόδους.
    - Αποφεύγεται η επανάληψη τυπικών εργασιών μέσα σε διαφορετικές μεθόδους (π.χ. η σχεδίαση μίας γραμμής ενός σχήματος)

3.2 Έλεγχος διαγραμμάτων ακολουθίας

- Ελέγχεται ότι τα διαγράμματα ακολουθίας περιγράφουν στη μεγαλύτερη δυνατή λεπτομέρεια τον τρόπο υλοποίησης του συστήματος, συμπεριλαμβάνοντας πρότυπα σχεδίασης (design patterns), καθώς και τα απαραίτητα στοιχεία της αρχιτεκτονικής (interfaces, libraries, components) και τον τρόπο με τον οποίο αυτά επηρεάζουν την υλοποίηση του συστήματος

3.2.1 Επιμέρους Έλεγχοι

3.2.1.1 Syntax Checks for Classes

- 1. Check class names.
- 2. Check class stereotypes.
- 3. Check the type of the class itself.
- 4. Check attributes. It is important to subject the attributes to syntax checks that are language specific.
- 5. Check attribute types.
- 6. Check attribute initial values. Ensure that the types of value initialization and the attribute types are compatible.
- 7. Check attribute visibility.
- 8. Check attribute stereotypes.
- 9. Check operations to ensure that their format compiles with the language of implementation.
- 10. Check operation signatures.
- 11. Check operation visibility.
- 12. Check operation stereotypes.

3.2.1.2 Semantic Checks for Classes

1. Check the meaning of the class.
2. Check the meanings of the attributes.
3. Check attribute initial values.
4. What does an operation mean? Ensure that the meaning of the operation is reflected in its name and format.
5. Check the pre- and postconditions of operations.
6. Check the signature of the operation.
7. Check the stereotypes of operations.
8. Check the scope of operations.
9. Check to see if the operations of a class are overloaded.
10. If overriding operations exist, ensure their correctness.
11. Check for overriding variables.
12. Check for encapsulation.

3.2.1.3 Syntax Checks for Class Diagrams

1. Check that the multiplicity on an association is correctly represented on the class diagram.
2. Ensure that stereotypes are represented correctly on classes, attributes, operations and relationships on a class diagram.
3. Check the association of classes with language libraries.
4. Check to see if the class is an exception class.
5. Check how error handling is modeled and implemented in the class.

3.2.1.4 Semantic Checks for Class Diagrams

1. Check directions of association.
2. Check the meaning of the relationships on a class diagram.
3. Check for collection classes.
4. Check the roles of classes.
5. Check the business rules behind the multiplicity.
6. Check for association classes.
7. Check if the operations of a class that has been specialized (inherited from) are overloaded.
8. Check for encapsulation.
9. Ensure that language constructs subject to interpretations are checked for their implied meaning.

3.2.1.5 Aesthetic Checks for Class Diagrams

1. Ensure that technical classes are represented only by their names rather than by their entire qualifications.
2. Improve the aesthetics by letting the entity classes appear in more than one diagram.
3. Improve the aesthetics by redistributing the classes and their associations across more than one class diagram.
4. Ensure that sufficient explanatory notes are provided.

3.2.1.6 Syntax Checks for Sequence Diagrams

1. Check the correctness of all objects on the sequence diagram.
2. Check the correctness of actors on the sequence diagram.
3. Check object-object interaction.
4. Check the message types shown in the sequence diagram.
5. Check the syntax of the message signatures and return values.

6. Check the syntax of multiple messages.

7. Check for multiple objects on the sequence diagram.

3.2.1.7 Semantic Checks for Sequence Diagrams

1. Check the meaning behind the sequence diagram.

2. Check the meaning behind the focus of control.

3. Check to see if the sequence diagram depicts creation and destruction of objects.

4. Check to see if the sequence diagram is based on a pattern.

5. Check to see if there are alternative flows and create separate sequence diagrams for them.

3.2.1.8 Aesthetic Checks for Sequence Diagrams

- 1. Ensure that the sequence diagram shows a cohesive set of interactions between collaborating objects.
- 2. Ensure that the sequence diagrams have sufficient notes and other annotations to explain the technicality of the diagrams.
- 3. Check the number of objects.
- 4. Check the number of messages