# Τεχνολογία Λογισμικού

**Ενότητα #11**: Ευέλικτες Μέθοδοι και Ακραίος Προγραμματισμός

Σταμέλος Ιωάννης
Τμήμα Πληροφορικής

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

# 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 την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- 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.

# Ευέλικτες Μέθοδοι και Ακραίος Προγραμματισμός

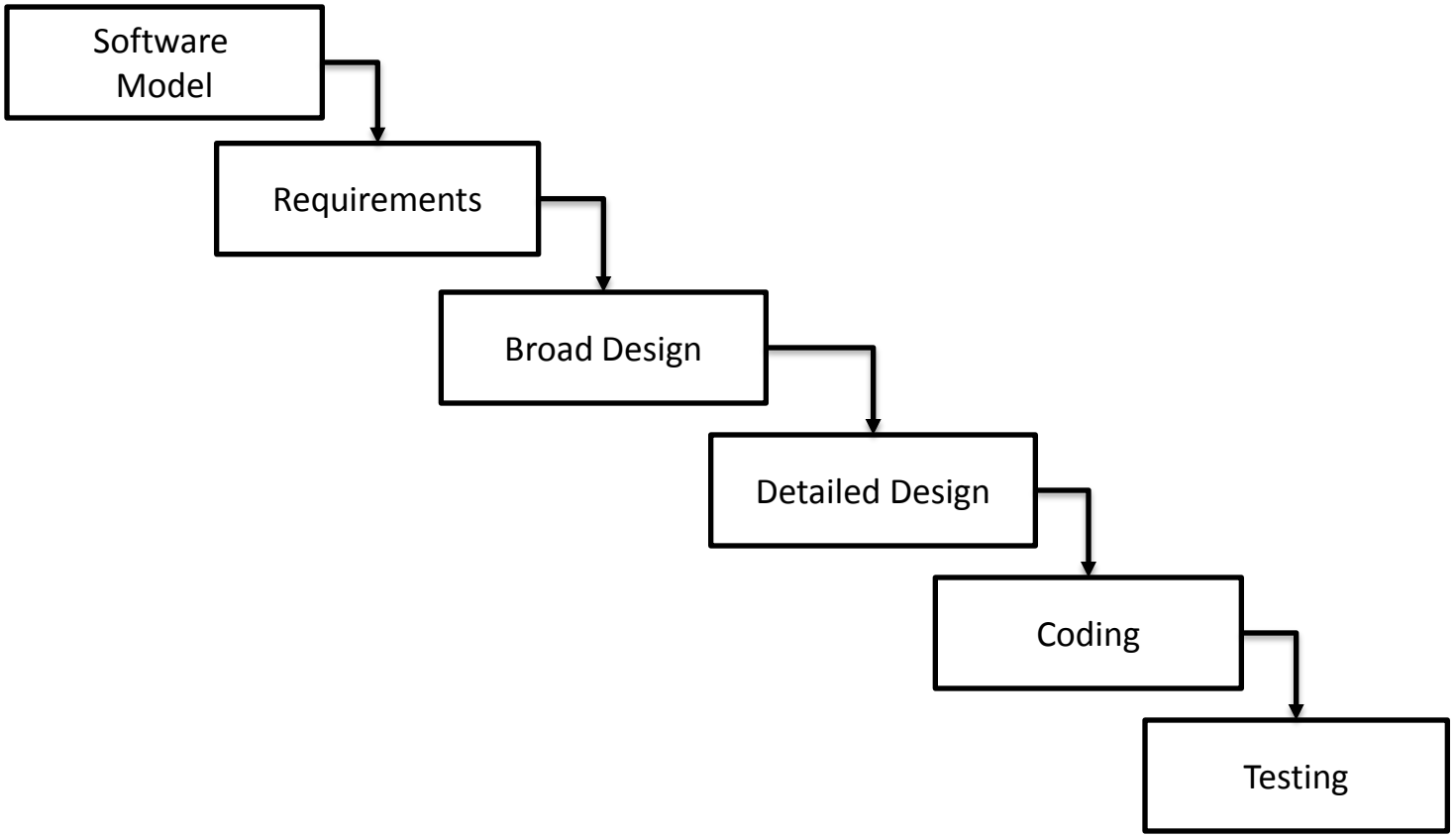
# Περιεχόμενα ενότητας

1. Πρότυπα προγραμματισμού.
2. Μοντέλο Καταρράκτη.
3. Ευέλικτες Μέθοδοι.
4. Ακραίος Προγραμματισμός.
5. Πρότυπα κωδικοποίησης.

# Πρότυπα και διαδικασίες προγραμματισμού

- Πρότυπα για εσάς (Fairley 1985).
- Πρότυπα για τους άλλους.
- Ταίριασμα σχεδίασης και υλοποίησης.

# Μοντέλο Καταρράκτη (Waterfall Model)




Κύκλος Ζωής – Ανάπτυξη Λογισμικού

# Μοντέλο Καταρράκτη – Μειονεκτήματα -1-

- Σειριακή επεξεργασία του έργου.
- Μη σωστός προσδιορισμός απαιτήσεων. Ο πελάτης γνωρίζει αργότερα τι ακριβώς θέλει.
- Τα λάθη ανακαλύπτονται αργά. Αύξηση κόστους διόρθωσης.
- Η πρώτη έκδοση έτοιμη πολύ αργά στον κύκλο ζωής.
- Έξοδα συντήρησης 70% των εξόδων συστήματος.

# Μοντέλο Καταρράκτη – Μειονεκτήματα -2-

- Βασικό πρόβλημα στην ανάπτυξη λογισμικού είναι οι κίνδυνοι.
- Παραδείγματα κινδύνων
  - Χρονοδιάγραμμα
  - Ακύρωση Έργου
  - “Λοξοδρόμηση” Συστήματος
  - Πλήθος λαθών
  - Παρανόηση Δραστηριότητας
  - Αλλαγή Δραστηριότητας
  - Λάθη σε μελλοντικές εκτιμήσεις
  - Αναδιοργάνωση προσωπικού.
- Τα “παραδοσιακά” μοντέλα δεν μπορούν να αντιμετωπίσουν τέτοιους κινδύνους.

# Ευέλικτες Μέθοδοι (Agile Methods)

- **Ευελιξία στον προγραμματισμό (Agility)** είναι η ικανότητα της **προσαρμογής** και **επαναπροσδιορισμού** ενός αναπτυσσόμενου και συνεχώς εξελισσόμενου συστήματος στην περίπτωση που εμφανίζονται αλλαγές στις αρχικές θεωρήσεις και παραδοχές.
- Οι Ευέλικτες μέθοδοι είναι:
  - Επαναληπτικές (Iterative)
  - Επαυξητικές (Incremental)
  - Αυτό-διοργανούμενες (Self-Organizing)
  - Προκύπτουσες (Emergent).

# Agile Manifesto -1-

- **Άτομα και Αλληλεπιδράσεις** αντί διαδικασίες και εργαλεία
- **Δυναμικός Κώδικας** αντί γραπτής τεκμηρίωσης
- **Συνεργασία με τον Πελάτη** αντί αυστηρών συμβολαίων
- **Ανταπόκριση σε αλλαγές** αντί ακολουθούμενου σχεδίου

- Ενώ υπάρχει αξία στα στοιχεία δεξιά (μετά το αντί), εμείς δίνουμε περισσότερη αξία στα στοιχεία αριστερά.



# Agile Manifesto -2-

- Αρχές των Ευέλικτων Μεθόδων:
  - Ικανοποίηση του πελάτη.
  - Συχνή παράδοση λογισμικού.
  - Η αλλαγή είναι ευπρόσδεκτη.
  - Καθημερινή συνεργασία με πελάτες.
  - Ικανό προσωπικό και περιβάλλον εμπιστοσύνης στην ομάδα.
  - Διαπροσωπική συζήτηση για την ανταλλαγή πληροφοριών.
  - Σωστή λειτουργία του λογισμικού που κατασκευάζεται.
  - Εξασφάλιση σταθερού ρυθμού ανάπτυξης.
  - Τεχνική αρτιότητα και καλός σχεδιασμός.
  - Υλοποίηση στόχων με σύντομο και αποτελεσματικό τρόπο.
  - Αυτό-διοργανονώμενες ομάδες.
  - Επαναπροσδιορισμός της συμπεριφοράς της ομάδας.

# Διαθέσιμες Ευέλικτες Μέθοδοι

- Agile Modeling.
- Adaptive Software Development (ASD).
- Crystal methods.
- Dynamic System Development Methodology (DSDM).
- eXtreme Programming (XP).
- Feature Driven Development (FDD).
- Lean Development.
- Scrum.

# Ακραίος Προγραμματισμός (XP - programming)

- Ένας ελαφρύς, αποτελεσματικός, χαμηλού-κινδύνου, ευέλικτος, προβλέψιμος, επιστημονικός και ευχάριστος τρόπος για την ανάπτυξη λογισμικού.
- Βασίζεται σε τέσσερις αξίες στην απλότητα, επικοινωνία, ανατροφοδότηση και κουράγιο. Η αποτελεσματικότητα της οφείλεται στη στενή συνεργασία της ομάδας κάτω από απλές πρακτικές με συχνή ανατροφοδότηση που τους επιτρέπει να αξιολογούν την πρόοδο τους και να προσαρμόζουν τις πρακτικές στις τρέχουσες ανάγκες.
- Το πρώτο XP-πρόγραμμα ήταν το πρόγραμμα μισθοδοσίας στην Chrysler Comprehensive Compensation (C3), (Beck – Highsmith, 1998).
- Γιατί ονομάστηκε Ακραίος Προγραμματισμός;

# Οι τέσσερις αρχές - (Values)

- Επικοινωνία – Communication.
- Απλότητα – Simplicity.
- Ανατροφοδότηση- Feedback.
- Κουράγιο – Courage.
- Επικοινωνία
  - Η κοινή κατανόηση των προβλημάτων του λογισμικού απαιτεί την διαπροσωπική επικοινωνία. Οτιδήποτε εμποδίζει την αμεσότητα αυτή πρέπει να αποβληθεί.

# Επικοινωνία - (συνέχεια..)

- Οι ομάδες προγραμματισμού XP:
  - Χρησιμοποιούν μια κοινή Αρχιτεκτονική εικόνα του συστήματος.
  - Εργάζονται σε ανοικτό χώρο εργασίας.
  - Διαρκώς ολοκληρώνουν τον κώδικα.
  - Επικοινωνούν με ένα πελάτη που βρίσκεται διαρκώς μαζί τους.
  - Προγραμματίζουν σε ζεύγη.
  - Κατέχουν όλοι τον κώδικα.
  - Διαρκώς σχεδιάζουν περιπτώσεις ελέγχου.

# Απλότητα - (Simplicity)

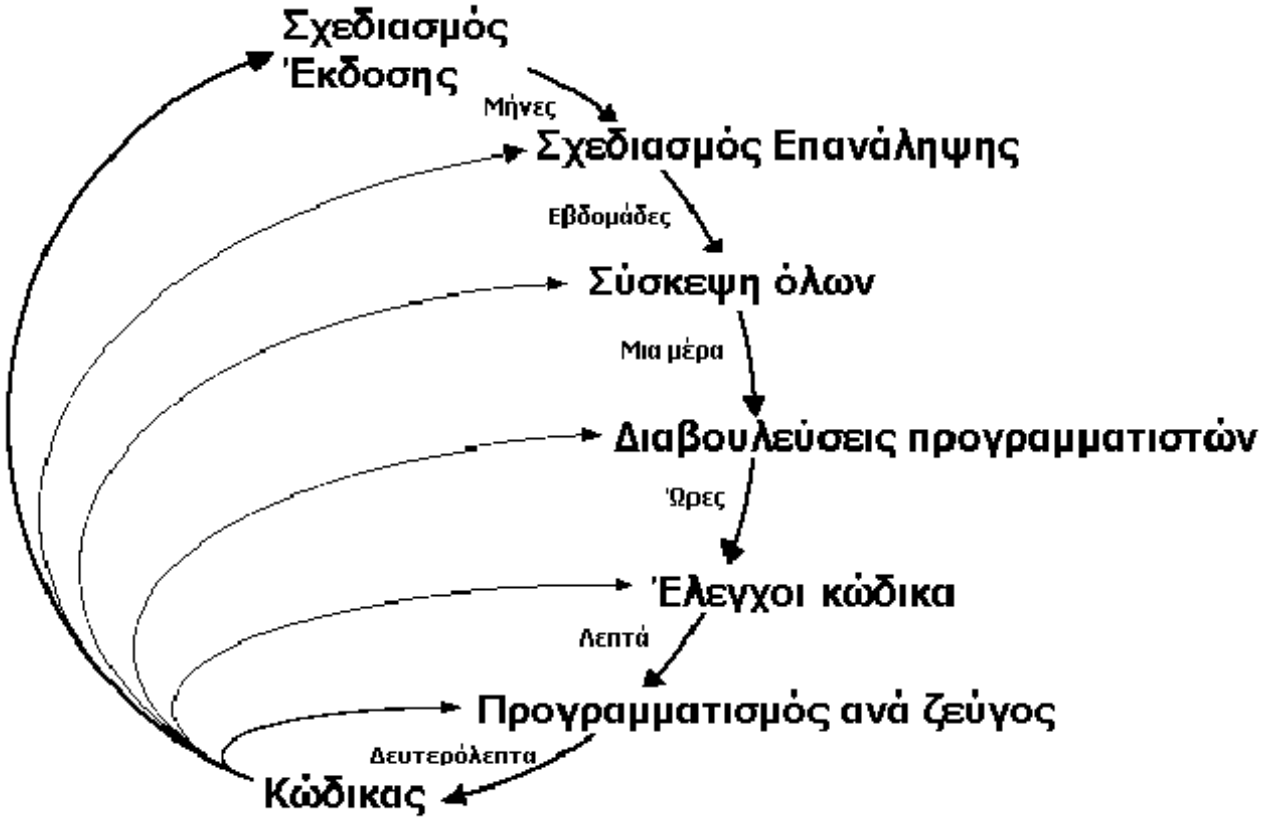
- Οι ομάδες προγραμματισμού ΧΡ:
  - Εκτελούν το πιο απλό σχέδιο που πιθανόν θα δουλέψει.
  - Συνεχώς απλοποιούν και βελτιώνουν την ανάπτυξη του κώδικα με αναδόμηση.

# Ανατροφοδότηση - (Feedback)

- Συγγραφή και χρήση Test cases πριν την παραγωγή κώδικα.
- Ανάπτυξη σε μικρές εκδόσεις και σε μικρότερες επαναλήψεις και σε μικρότερες εργασίες και σε ακόμη μικρότερα tests.

.....

# Ανατροφοδότηση - (Feedback) (2)



# Κουράγιο - (Courage)

- Τα μέλη της ομάδας ΧΡ δεν φοβούνται να:
  - Σταματούν όταν κουράζονται.
  - Να αφήνουν τις οικονομικές αποφάσεις στους πελάτες.
  - Προτείνουν στους πελάτες να αλλάξουν την εμβέλεια μιας έκδοσης.
  - Να ζητούν βοήθεια όταν χρειάζεται.
  - Y A G N I (You’re not gonna need it!).
  - Αλλάξουν την σχεδίαση και τον κώδικα.
  - Πετάξουν κώδικα που δεν ικανοποιεί.
  - Αλλάξουν την διαδικασία ανάπτυξης όταν δεν λειτουργεί.

# Οι 12 – πρακτικές (XP - practices)

- Το παιχνίδι του σχεδιασμού - The Planning Game
- Μικρές εκδόσεις - Small releases
- Αρχιτεκτονική εικόνα - Metaphor
- Απλή Σχεδίαση - Simple design
- Έλεγχοι πριν την κωδικοποίηση - Testing
- Ανακατασκευή κώδικα - Refactoring
- Προγραμματισμός ανά ζεύγη - Pair Programming
- Συλλογική ιδιοκτησία κώδικα - Collective Ownership
- Διαρκείς ενοποιήσεις του κώδικα - Continuous Integration
- Υποφερτός ρυθμός εργασίας - Sustainable pace
- Διαρκής παρουσία Πελάτη - On-site customer
- Πρότυπα κωδικοποίησης - Coding standards

# Οι πρακτικές σε εικόνες...

| | | | |
|---|---|---|---|
| Το παιχνίδι του σχεδιασμού<br>  <br>=συμβολή πελάτη | Προγραμματισμός ανά ζεύγη<br>  <br>=λιγότερα λάθη στον κώδικα | Έλεγχοι πριν την κωδικοποίηση<br>  <br>=ποιοτικό λογισμικό | Ανακατασκευή κώδικα<br>  <br>=καθαρός κώδικας |
| Σταθερές κωδικοποίησης<br>  <br>=καλύτερη επικοινωνία | Απλή σχεδίαση<br>  <br>=ευελιξία | Μικρές εκδόσεις<br>  <br>=προσαρμοστικότητα | Διαρκείς ενοποιήσεις του κώδικα<br>  <br>=διαρκής ανάδραση |
| Συλλογική ιδιοκτησία κώδικα<br>  <br>=διαμοιρασμός γνώσης | Διακρή παρουσία πελάτη<br>  <br>=ξεκαθάρισμα απαιτήσεων | Αρχιτεκτονική εικόνα<br>  <br>=κατανόηση συστήματος | Υποφερτός ρυθμός εργασίας<br>  <br>=αποδοτικότητα |

# Το παιχνίδι του σχεδιασμού (Planning Game)

- Το παιχνίδι του σχεδιασμού είναι μία δραστηριότητα με την οποία η ομάδα ανάπτυξης του συστήματος και οι πελάτες αποφασίζουν τι θα γίνει σε κάθε έκδοση – release (3-6 μήνες) και κάθε επανάληψη – iteration (1-3 εβδομάδες).
- Το παιχνίδι του σχεδιασμού (planning game) τρέχει για κάθε επανάληψη για να καθορίσει ποια λειτουργία θα δρομολογηθεί στην επόμενη ενσωμάτωση. Οι προγραμματιστές λαμβάνουν τις τεχνικές αποφάσεις - εκτιμήσεις και οι πελάτες τις επιχειρησιακές αποφάσεις.

# Προγραμματισμός ανά ζεύγη (Pair Programming)

- Η ανάπτυξη του προγράμματος βασίζεται πάντα σε δύο άτομα που μοιράζονται τον ίδιο υπολογιστή.
- Συνήθως ο ένας γράφει ενώ ο άλλος βλέπει, διορθώνει και σκέφτεται ένα βήμα μπροστά.
- Τα ζευγάρια εναλλάσσονται συνεχώς με αποτέλεσμα να μεταφέρεται η εμπειρία και η γνώση σε όλα τα μέλη της ομάδας.
- Η πιο βασική πρακτική μαζί με την πρακτική των ελέγχων πριν την κωδικοποίηση.

# Έλεγχοι πριν την κωδικοποίηση (Test-first-design)

- Οι προγραμματιστές γράφουν περιπτώσεις τεστ πριν αρχίσουν τη συγγραφή κώδικα.
- Η ομάδα δημιουργεί αυτοματοποιημένα τεστ μονάδας – **unit tests** και τεστ αποδοχής - **acceptance tests** τα οποία εφαρμόζονται συχνά.
- Το πρόγραμμα ελέγχεται κάθε φόρα που προστίθεται επιπλέον κώδικας.
- Για κάθε κομμάτι κώδικα δημιουργείται αντίστοιχο τεστ.
- Τα αυτοματοποιημένα τεστ τρέχουν σε όλο το πρόγραμμα και διασφαλίζουν ότι όλα λειτουργούν σωστά.

# Ανακατασκευή κώδικα (Refactoring)

- Η τεχνική βελτίωσης του υπάρχοντος κώδικα δίχως να μεταβληθεί η λειτουργικότητά του.
- Ο κώδικας απλοποιείται και γίνεται πιο ευέλικτος και κατανοητός.
- Μελλοντικές αλλαγές ή προσθήκες είναι εύκολα υλοποιήσιμες.

# Πρότυπα κωδικοποίησης (Coding standards)

- Η συγγραφή κώδικα είναι μία ομαδική εργασία. Κατά καιρούς διαφορετικά άτομα θα εργαστούν σε διαφορετικά τμήματα κώδικα. Οι διαφορές στο ύφος καθιστούν συχνά τον κώδικα δύσκολο αντικείμενο εργασίας.
- Ο κώδικας συχνά αναδομείται και η αρχιτεκτονική των συστημάτων αλλάζει. Προκειμένου να υπάρχει αποτελεσματικότητα πρέπει ο κώδικας όλης της ομάδας να μοιάζει σαν να γράφτηκε από ένα μόνο άτομο και για να επιτευχθεί αυτό απαιτούνται πρότυπα κώδικα και σταθερές κωδικοποίησης.

# Απλή Σχεδίαση (Simple design)

- Ο σωστός σχεδιασμός μίας εφαρμογής πρέπει:
  - να “τρέχει” σε όλα τα test
  - να έχει απλή λογική
  - να δηλώνει κάθε πρόθεση σημαντική για τους προγραμματιστές
  - να έχει τις λιγότερες δυνατές κλάσεις και μεθόδους.

# Μικρές εκδόσεις (Small releases)

- Οι εκδόσεις πρέπει να είναι όσο το δυνατόν μικρότερες. Κάθε έκδοση περιέχει μόνο τα πιο σημαντικά χαρακτηριστικά (αυτά που έχουν συμφωνηθεί).
- Οι XP ομάδες πρέπει να δίνουν εκδόσεις στο τέλος κύκλων έκδοσης. Ο κύκλος έκδοσης πρέπει να είναι όσο το δυνατόν μικρότερος, χωρίς όμως να υπάρχουν χαρακτηριστικά που δεν δουλεύουν απόλυτα.
- Πριν την έκδοση υλοποιείται τεστ-αποσφαλμάτωσης και μετά γίνεται η ενσωμάτωση του κώδικα.

# Διαρκείς ενοποιήσεις του κώδικα (Continuous Integration)

- Οι XP ομάδες εργάζονται με μικρά βήματα και ενσωματώνουν τον κώδικά τους αρκετές φορές την ημέρα. Αυτό σημαίνει πως προβλήματα ενσωμάτωσης ανακαλύπτονται γρήγορα από τη στιγμή που εμφανιστούν και είναι πιο εύκολο να διευθετηθούν.
- Η συνεχής ενσωμάτωση κώδικα στο πρόγραμμα συνεπάγεται ότι δεν υπάρχουν μεγάλες εξελίξεις που είναι ασυμβίβαστες με το υπόλοιπο πρόγραμμα και ότι όλοι μπορούν να εργάζονται πάνω στην πιο πρόσφατη έκδοση του συστήματος.

# Συλλογική ιδιοκτησία κώδικα (Collective Ownership)

- Καθένας στην ομάδα έχει την δικαιοδοσία να αλλάξει οτιδήποτε στον κώδικα, αρκεί να κάνει τις αλλαγές με τον συνάδελφό του, να ακολουθούν τα πρότυπα κώδικα, και να διασφαλίσουν ότι όλα τα τεστ δουλεύουν, όταν τελειώσουν τις αλλαγές.
- Αυτό αφαιρεί τις δυσχέρειες και τις αρχιτεκτονικές διαστρεβλώσεις που μπορούν να εμφανιστούν με τη μεμονωμένη-προσωπική ιδιοκτησία κώδικα.

# Διαρκής παρουσία Πελάτη (On-site customer)

- Καμία γραπτή απαίτηση δεν είναι πλήρης και σαφής. Οι προγραμματιστές χρειάζονται πάντα επικοινωνία με τον πελάτη για διευκρινίσεις, ανεξάρτητα από το πόση προσπάθεια καταβλήθηκε στην αρχική προδιαγραφή απαιτήσεων.
- Μία XP ομάδα παρακάμπτει όλη αυτή την προσπάθεια αποτελεσματικής προδιαγραφής και ανάλυσης απαιτήσεων έχοντας κάποιον πελάτη διαθέσιμο συνέχεια στο χώρο εργασίας.

# Αρχιτεκτονική εικόνα (Metaphor)

- Η αρχιτεκτονική εικόνα που δίνει συνοχή και συνέπεια στον τρόπο με τον οποίο η ομάδα αναπτύσσει το σύστημα.
- Μέσα στις ιδιότητες της αρχιτεκτονικής εικόνας περιλαμβάνεται και η ονομασία των διαφόρων κλάσεων και μεθόδων. Είναι πολύ σημαντική η ονοματολογία στην κατανόηση της αρχιτεκτονικής του συστήματος και στη δυνατότητα επαναχρησιμοποίησης κώδικα.

# Υποφερτός ρυθμός εργασίας (Sustainable pace)

- Η ανάπτυξη λογισμικού είναι μία δημιουργική εργασία και κανείς δεν μπορεί να είναι παραγωγικός και δημιουργικός αν είναι εξαντλημένος.
- Περιορίζοντας τις ώρες εργασίες σε 40-ώρες ανά εβδομάδα διατηρεί την ομάδα ξεκούραστη, μειώνει τον κύκλο εργασιών του προσωπικού, και βελτιώνει την ποιότητα του ολοκληρωμένου προϊόντος.

# Ακραίος Προγραμματισμός…

- Είναι μια πειθαρχημένη προσέγγιση όπου πρέπει να:
  - Γράφετε test πριν τον κώδικα.
  - Προγραμματίζετε σε ζεύγη.
  - Ολοκληρώνετε τακτικά τον κώδικα.
  - Ξεκουράζονται οι προγραμματιστές.
  - Επικοινωνούν οι προγραμματιστές με τους πελάτες συνεχώς στο χώρο εργασίας.
  - Ακολουθούνται οι προτεραιότητες των πελατών.
  - Αφήνεται το λογισμικό καθαρό και απλό στο τέλος της ημέρας.
  - Προσαρμόζεστε στις διαδικασίες και πρακτικές του περιβάλλοντος σας.

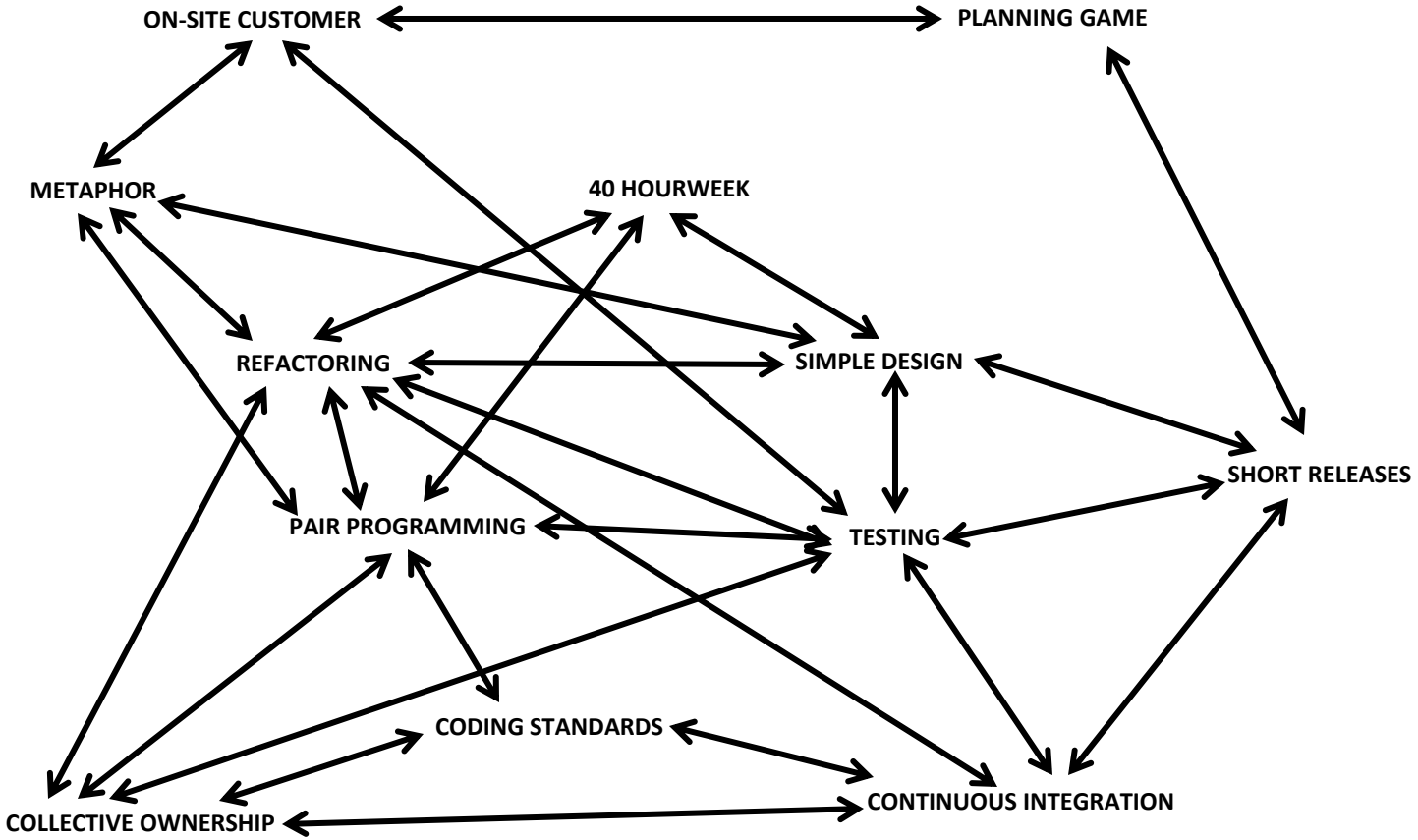
# Γιατί στα άκρα;

- Αν η **επιθεώρηση κώδικα** είναι ωφέλιμη, τότε να κάνετε συνεχώς επιθεωρήσεις (**pair programming**).
- Αν ο **έλεγχος κώδικα** είναι ωφέλιμος, τότε ελέγχετε συνεχώς (**unit tests - acceptance tests**).
- Αν ο **ανασχεδιασμός είναι καλός**, τότε ανασχεδιάζετε συνεχώς (**refactoring**).
- Αν η **απλότητα είναι καλή**, τότε κάνε το απλούστερο που μπορεί να δουλέψει (**simple design**).

# Γιατί στα άκρα;

- Αν η **αρχιτεκτονική του συστήματος** είναι σημαντική, τότε όλοι θα την ορίζουν και επανακαθορίζουν (**metaphor**).
- Αν οι **έλεγχοι ολοκλήρωσης** είναι σημαντικοί, τότε να εκτελείς τέτοιους ελέγχους καθημερινά (**continuous integration**).
- Αν η **ανατροφοδότηση είναι καλή**, τότε να την λαμβάνεις συνεχώς (**pair programming, planning game, on-site customer**).

# Αλληλεπίδραση πρακτικών-1



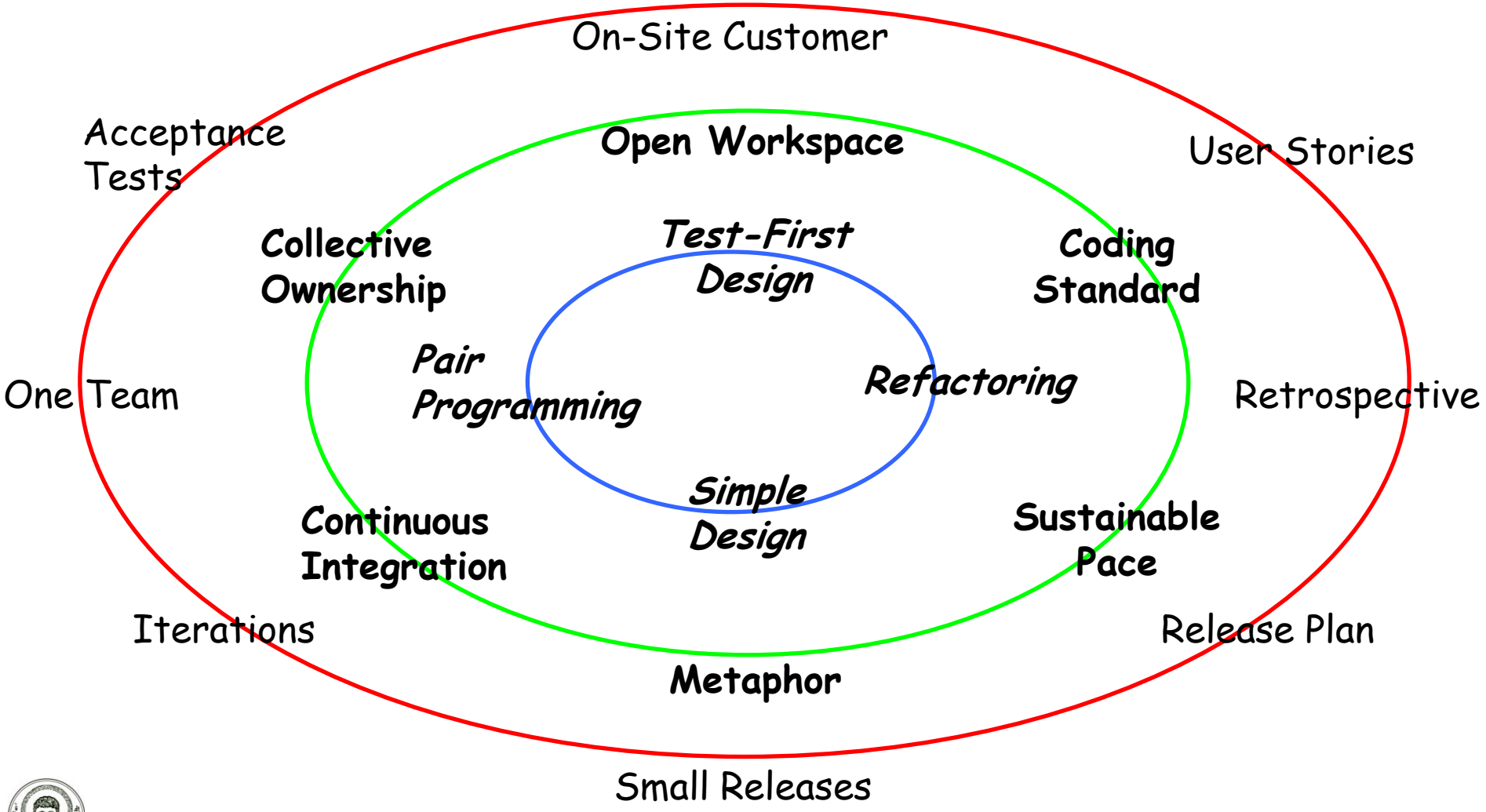
# Αλληλεπίδραση πρακτικών-2

Testing + Refactoring = ασφάλεια κώδικα

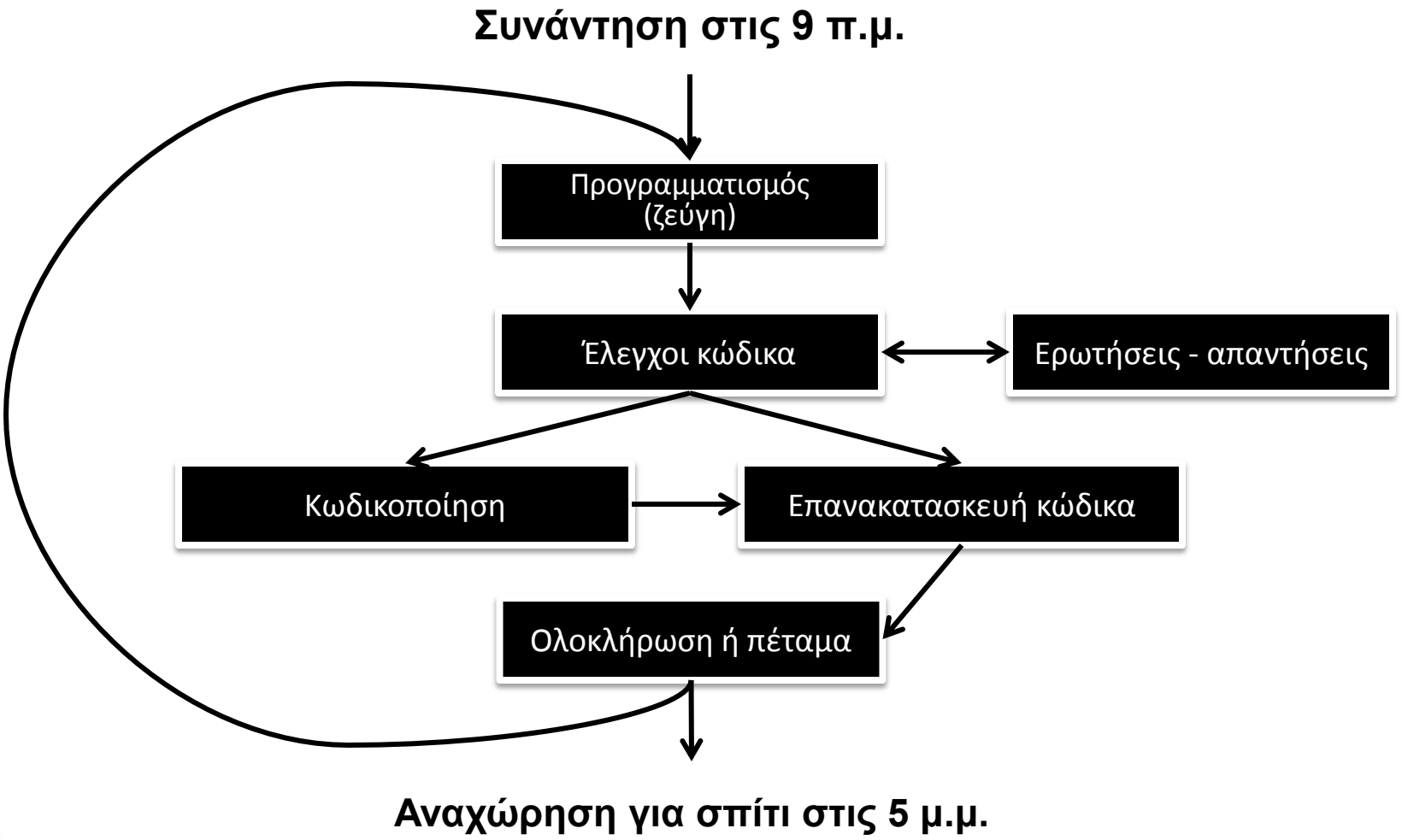
Simple design + Refactoring = απλότητα κώδικα

Pair Programming + Sustainable pace = παραγωγικότητα

# Οι κύκλοι λειτουργίας



# Κύκλος ανάπτυξης

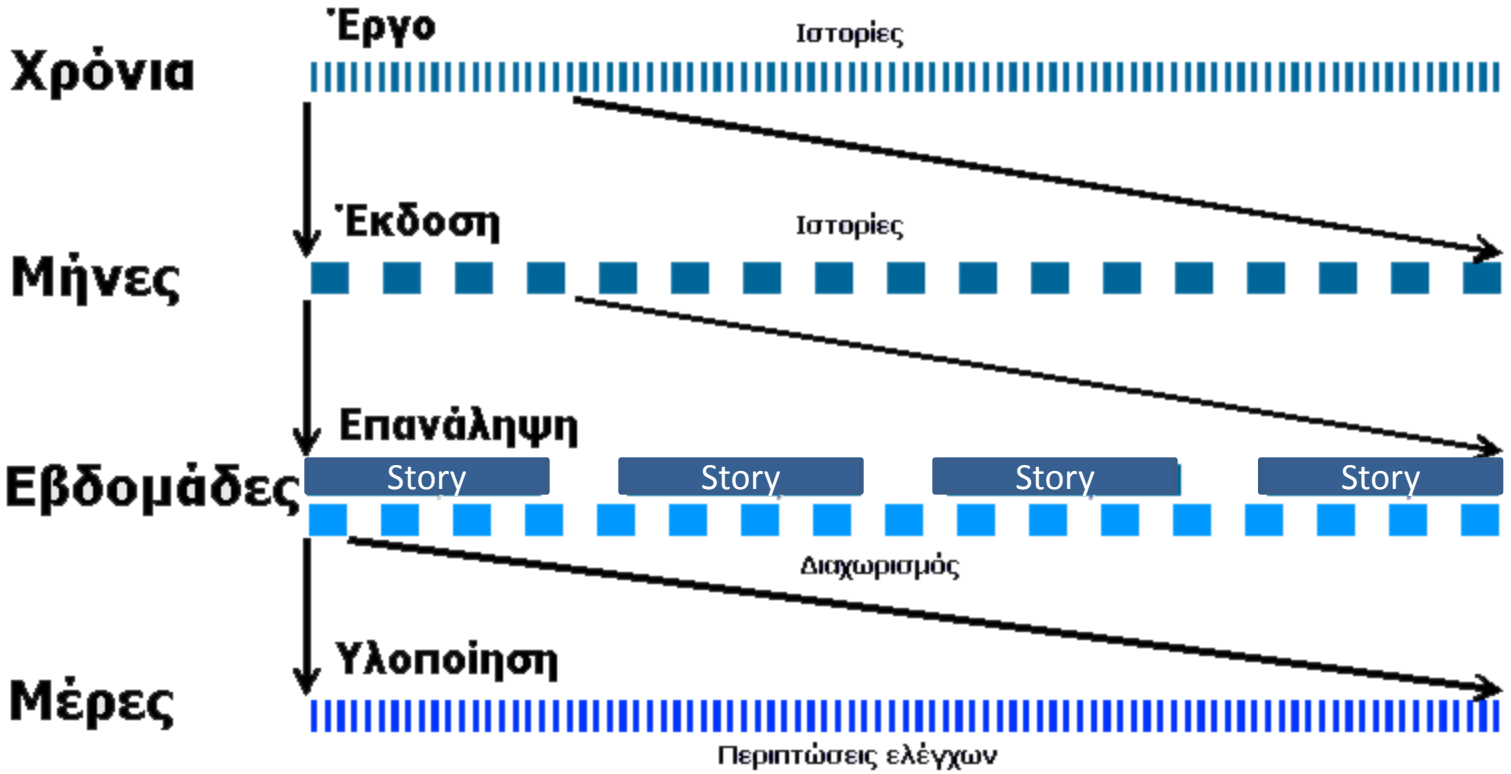


# Η ιεραρχία του παιχνιδιού σχεδιασμού

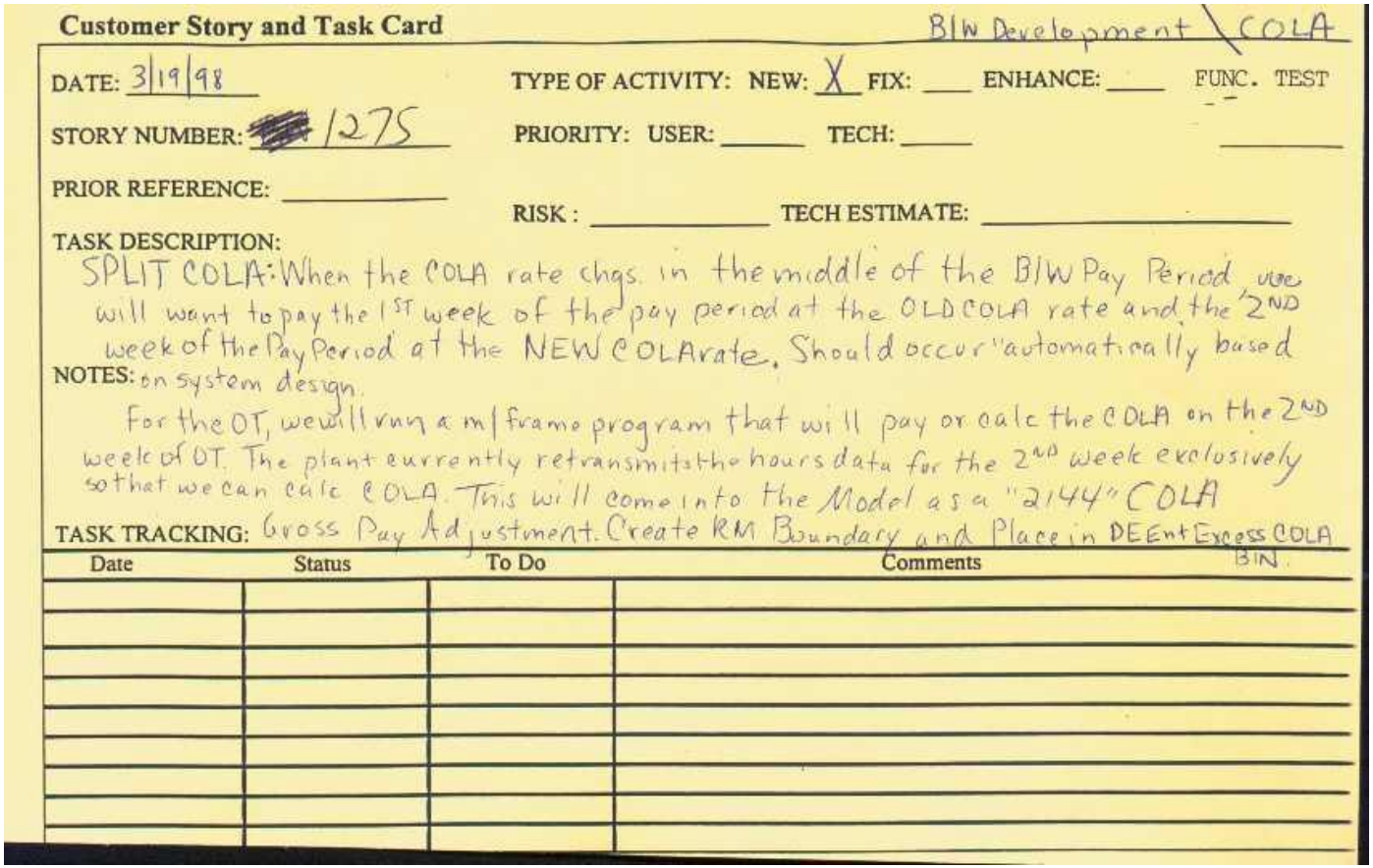
- Ένα project αποτελείται από **πολλές ιστορίες**

  Οι ιστορίες θα αλλάξουν σύντομα.
- Το project σχεδιάζεται σε **εκδόσεις (releases) των 2-3 μηνών.**
- Κάθε έκδοση σχεδιάζεται σε **επαναλήψεις (iterations) των 2-3 εβδομάδων.**
- Κάθε επανάληψη σχεδιάζεται **σε κομμάτια εργασιών (tasks) 1-2 ημερών.**
- Κομμάτια εργασιών σχεδιάζονται σε **περιπτώσεις τεστ (test cases)** που απαιτούν 5-10 λεπτά για να αναπτυχθούν.

# Η διαδικασία ανάπτυξης σχηματικά…



# Κάρτα Ιστορίας (story card -1)



**Customer Story and Task Card** — B1W Development \ COLA

DATE: 3/19/98 — TYPE OF ACTIVITY: NEW: X FIX: ___ ENHANCE: ___ FUNC. TEST ___

STORY NUMBER: [redacted]1275 — PRIORITY: USER: ___ TECH: ___

PRIOR REFERENCE: ___ — RISK: ___ TECH ESTIMATE: ___

TASK DESCRIPTION:
SPLIT COLA: When the COLA rate chgs. in the middle of the B1W Pay Period we will want to pay the 1ST week of the pay period at the OLD COLA rate and the 2ND week of the Pay Period at the NEW COLA rate. Should occur "automatically based on system design.

NOTES:
For the OT, we will run a m/frame program that will pay or calc the COLA on the 2ND week of OT. The plant currently retransmits the hours data for the 2ND week exclusively so that we can calc COLA. This will come into the Model as a "2144" COLA

TASK TRACKING: Gross Pay Adjustment. Create RM Boundary and Place in DEEntExcessCOLA BIN.

| Date | Status | To Do | Comments |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

*Πηγή: [xprogramming.com/articles/story_and_task_cards/](xprogramming.com/articles/story_and_task_cards/)*

# Κάρτα Ιστορίας (story card -2)

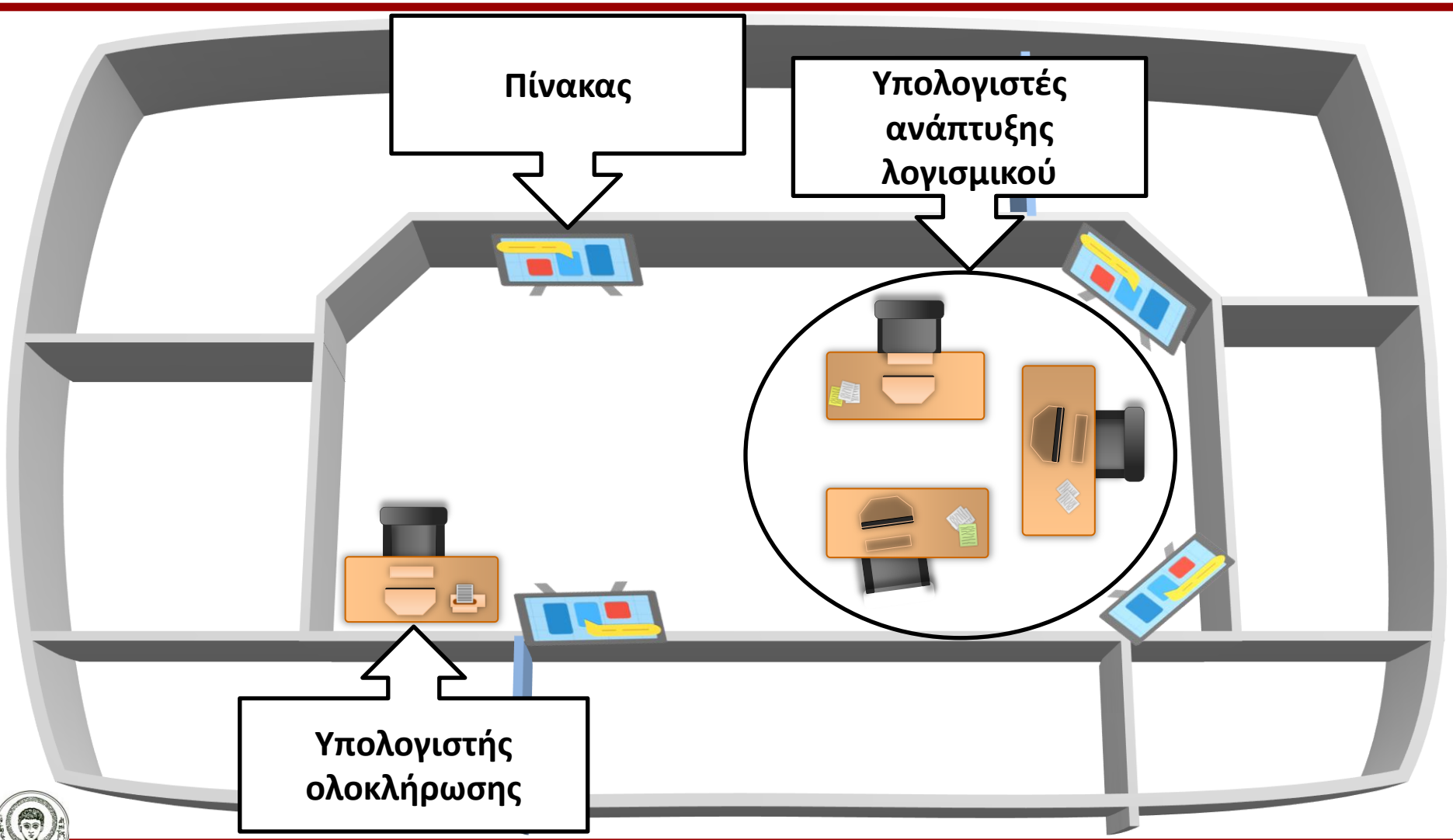
**101 Union Dues**

Bargaining Unit EEs have union dues withheld from the first pay period of the month. The amount varies by union, local, and in some cases varies by the individual.

If dues cannot be taken in the first pay period, they should not be taken until a UD30 transaction is received.

Priority: **High** Risk: **Low** Estimate: **1**

# Χώρος εργασίας



# Υπευθυνότητες πελάτη

- Ανάγκες (Needs)
- Ιστορίες (Stories)
- Πόροι (Resources)
- Προτεραιότητες (Priorities)
- Έγκριση (Acceptance)

# Υπευθυνότητες προγραμματιστή

- Εκτίμηση χρόνου (Time estimates)
- Σχεδιασμός (Design)
- Κωδικοποίηση (Code)
- Ποιότητα (Quality)

# Ο προπονητής της ομάδας (Coach)

- Είναι διαθέσιμος κάθε στιγμή για την επίλυση προβλημάτων, ιδίως για τους νέους προγραμματιστές.
- Προσέχει την εξέλιξη της συνολικής Επανα-κατασκευής κώδικα.
- Βοηθά στους ελέγχους, επανακατασκευή και οποιαδήποτε πρακτική του ζητηθεί.
- Δια-συνδέει τους προγραμματιστές με το διευθυντικό προσωπικό.

# Ο ιχνηλάτης (tracker)

- Μετρά τα πάντα. Καλά στα λόγια, αλλά στα έργα !!!
- Ανακαλύπτει και εφαρμόζει μετρικές.
- Ελαχιστοποιεί τις υπερβάσεις (χρόνου, χρήματος, κ.λ.π.).
- Δύο φορές την εβδομάδα τουλάχιστον ο έλεγχος μετρικών.

# Ο διευθύνων (manager)

- Μερικές από τις εργασίες που εκτελεί:
  - Χαράζει τις κατευθυντήριες γραμμές (πλάνο έκδοσης).
  - Βρίσκει λύσεις στα προβλήματα και παρεμβαίνει.
  - Χειρίζεται θέματα του προσωπικού (και νέες προσλήψεις).
  - Αλλάζει την διαδικασία ανάπτυξης (όταν χρειαστεί).
  - Ακυρώνει το έργο όταν δεν πάει καλά.
  - Ρυθμίζει τις συναντήσεις με τους συμμετέχοντες.
  - Διευθετεί όλα τα εργασιακά προβλήματα.

# CRC κάρτες - 1

- Class - Responsibility - Collaborator

  Wirfs-Brock: Responsibility-Driven Design
- Οι πελάτες και οι διευθυντές τις καταλαβαίνουν..
- **Αποσύνθεση των απαιτήσεων.**
- Γράφουμε κάθε κλάση σε ξεχωριστή κάρτα.
- Γράφουμε “**υπευθυνότητες**” (3-5).
- Δίπλα γράφουμε τις **συνεργαζόμενες κλάσεις.**
- Εύκολες για όλους αλλά δεν καταγράφονται μόνιμα.

# CRC κάρτες - 2

- Απαριθμούμε πρώτα τις υποψήφιες κλάσεις.
- Για κάθε μία κλάση βρίσκουμε το "**ουσιαστικό**" – όνομα της κλάσης π.χ. Sales, ATM, κ.λ.π.
- Μετά βρίσκουμε τα "**ρήματα ή ρήματα / επιρρήματα**" – υπευθυνότητες της κλάσης.
- Τέλος ορίζουμε πιθανές σχέσεις μεταξύ των αντικειμένων για τις συνεργασίες π.χ.
  - το αντικείμενο Α είναι σαν το αντικείμενο Β.
  - το αντικείμενο Α γνωρίζει το αντικείμενο Β.
  - το αντικείμενο Α μοιράζεται πληροφορίες με το αντικείμενο Β.

# CRC κάρτες - 3

## CRC – κάρτα

| Class name | |
|---|---|
| Subclasses: | |
| Superclasses: | |
| Responsibilities: | Collaborators |
| | |
| | |
| | |
| | |
| | |
| | |

# CRC κάρτες – Παράδειγμα 1ο

| Sale | |
|---|---|
| Add items to order | Sales Line Item |
| Determine total price | |
| Check for valid payment | Customer |
| Dispatch to delivery address | |

# CRC κάρτες – Παράδειγμα 2ο

| Class ATM | |
|---|---|
| **Responsibilities** | **Collaborators** |
| • Start up when switch is turned on<br>• Shut down when switch is turned off<br>• Start a new session when card is inserted by customer<br>• Provide access to component parts for sessions and transactions | • OperatorPanel<br>• CashDispenser<br>• NetworkToBank<br>• CustomerConsole<br>• Session |

# CRC κάρτες – Παράδειγμα 3ο

| Class CardReader | |
|---|---|
| **Responsibilities** | **Collaborators** |
| • Tell ATM when card is inserted<br>• Read information from card<br>• Eject card<br>• Retain card | • ATM<br>• Card |

# Έλεγχοι με τις κάρτες CRC-1

| CLASS<br>Elevator Controller |
|---|
| **RESPONSIBILITY** |
| 1. Turn on elevator button |
| 2. Turn off elevator button |
| 3. Turn on floor button |
| 4. Turn off floor button |
| 5. Move elevator up one floor |
| 6. Move elevator down one floor |
| 7. Open elevator doors and start timer |
| 8. Close elevator doors after timeout |
| 9. Check requests |
| 10. Update requests |
| **COLLABORATION** |
| 1. Class **Elevator Button** |
| 2. Class **Floor Button** |
| 3. Class **Elevator** |

- Σκεφτείτε την "**υπευθυνότητα**": **Turn on elevator button.**
- Δεν "**στέκει**" στον Αντικειμενοστρεφή Προγραμματισμό.
- Αγνοήσαμε την "*υπευθυνότητα*".
- Η υπευθυνότητα:

  Turn on elevator button

  Θα γίνει: **Send message to Elevator Button to turn itself on.**

# Έλεγχοι με τις κάρτες CRC-2

| CLASS<br>Elevator Controller |
|---|
| **RESPONSIBILITY** |
| 1. Send message to Elevator Button to turn on button |
| 2. Send message to Elevator Button to turn off button |
| 3. Send message to Floor Button to turn on button |
| 4. Send message to Floor Button to turn off button |
| 5. Send message to Elevator to move up one floor |
| 6. Send message to Elevator to move down one floor |
| 7. Send message to Elevator Doors to open |
| 8. Start timer |
| 9. Send message to Elevator Doors to close after timeout |
| 10. Check requests |
| 11. Update requests |
| **COLLABORATION** |
| 1. Subclass **Elevator Button** |
| 2. Subclass **Floor Button** |
| 3. Class **Elevator Doors** |
| **4. Class Elevator** |

- "Open elevator doors and start timer". Η κλάση "elevator" δεν έχει ορισθεί σωστά, γιατί:
- οι πόρτες έχουν **κατάσταση (state)** που αλλάζει δυναμικά με την εκτέλεση.
- Επομένως πρόσθεσε την κλάση "**elevator doors**".
- Πρόσεξε την ασφάλεια του συστήματος.
- Άρα: **Πρόσθεσε νέα κλάση, άλλαξε τα δυναμικά μοντέλα και τα μοντέλα των περιπτώσεων χρήσης.**

# Σημείωμα Χρήσης Έργων Τρίτων

Όλα τα σχήματα/διαγράμματα έχουν συμπεριληφθεί μετά από κατάλληλη τροποποίηση, από το σύγγραμμα «S. L. Pfleeger (Γ. Σταμέλος), «Τεχνολογία Λογισμικού, Θεωρία και Πράξη», Εκδ. ΚΛΕΙΔΑΡΙΘΜΟΣ, 2012.»

Highsmith J. , 1998. Adaptive Software Development: An Evolutionary Approach to Managing Complex Systems, New York: Dorset House.

Fairley R, (1985), Software engineering concepts, McGraw-Hill. ISBN:007066272X, 9780070662728

# Σημείωμα Αναφοράς

# Σημείωμα Αδειοδότησης

# Τέλος ενότητας

Επεξεργασία: <Τέγος Στέργιος >
Θεσσαλονίκη, <Χειμερινό Εξάμηνο 2013-2014>

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



# Σημειώματα

# Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.