Διαδικασίες Τεχνολογίας Λογισμικού

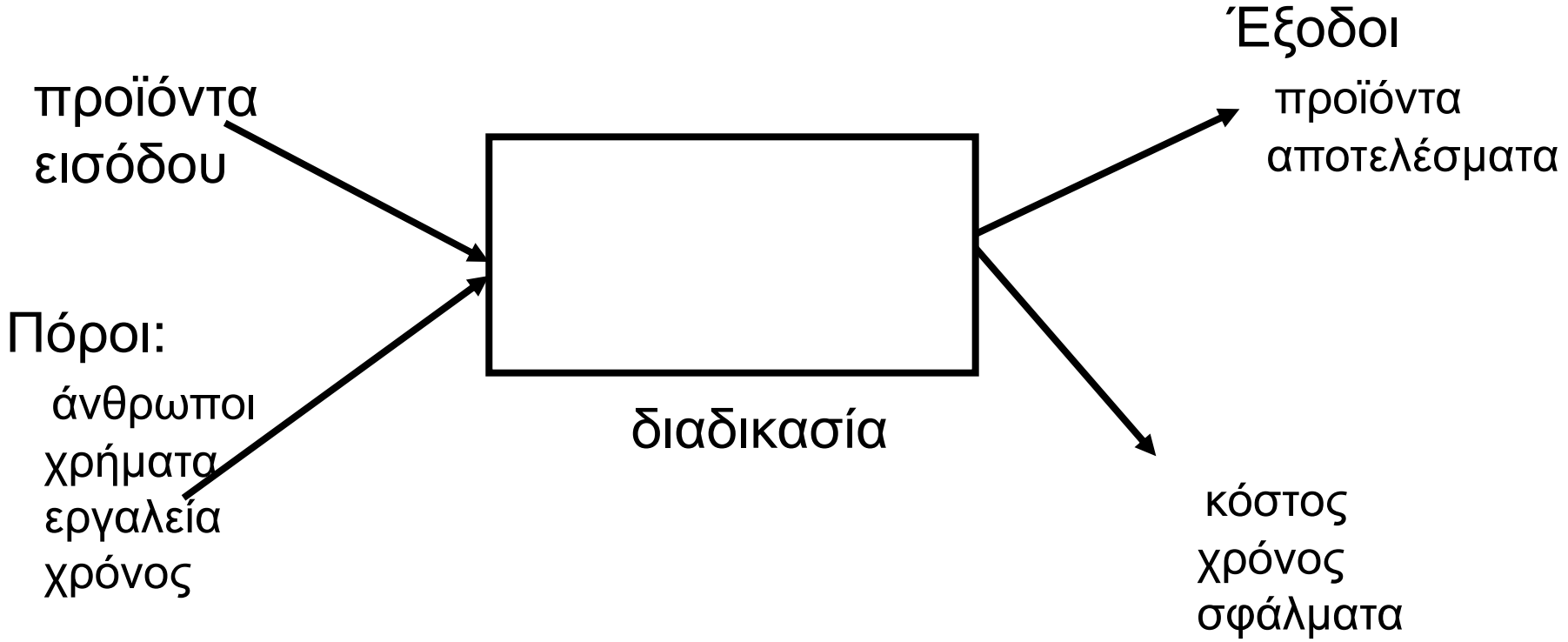
Γιάννης Σμαραγδάκης

Βασική έμφαση της τεχνολογίας λογισμικού

- **Πώς να περιγράψουμε προϊόντα;**
  - **τα συστατικά τους**
  - **τη σχέση και αλληλεπίδρασή τους**
- **Ποιες διαδικασίες πρέπει να ακολουθηθούν για να αναπτύξουμε τέτοια προϊόντα;**
  - **και να εξασφαλίσουμε την «ποιότητά» τους εν τέλει;**
  - **(παραδείγματα διαδικασίας;)**
- **Πώς να αναπτύξουμε και να εξελίξουμε προϊόντα με:**
  - **αποδεκτό κόστος**
  - **βελτιωμένη ποιότητα**

Τεχνολογία λογισμικού = Προϊόντα + Διαδικασίες

Η διαδικασία σαν απλή συνάρτηση

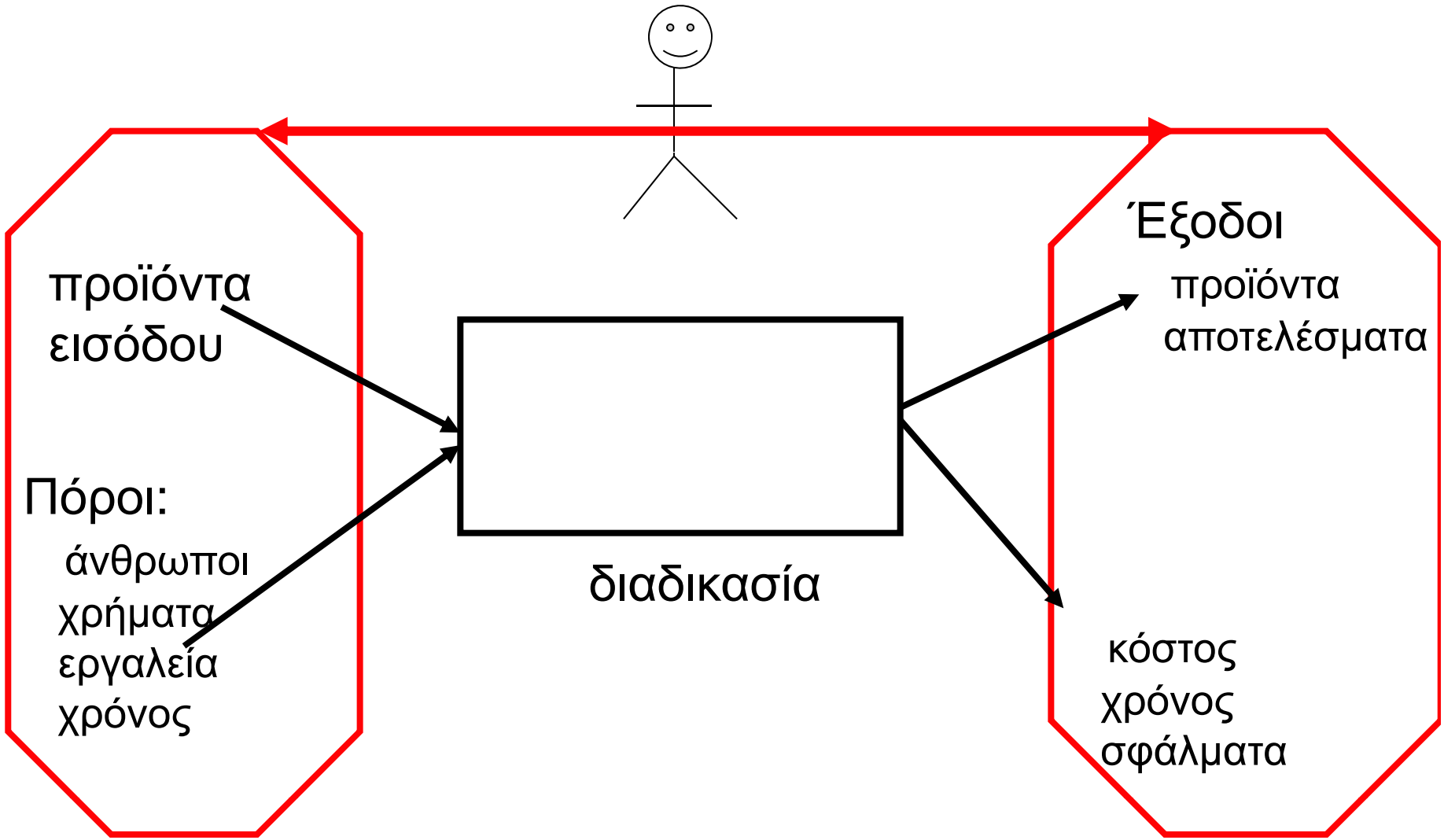


Δύο συμπληρωματικές θεωρήσεις

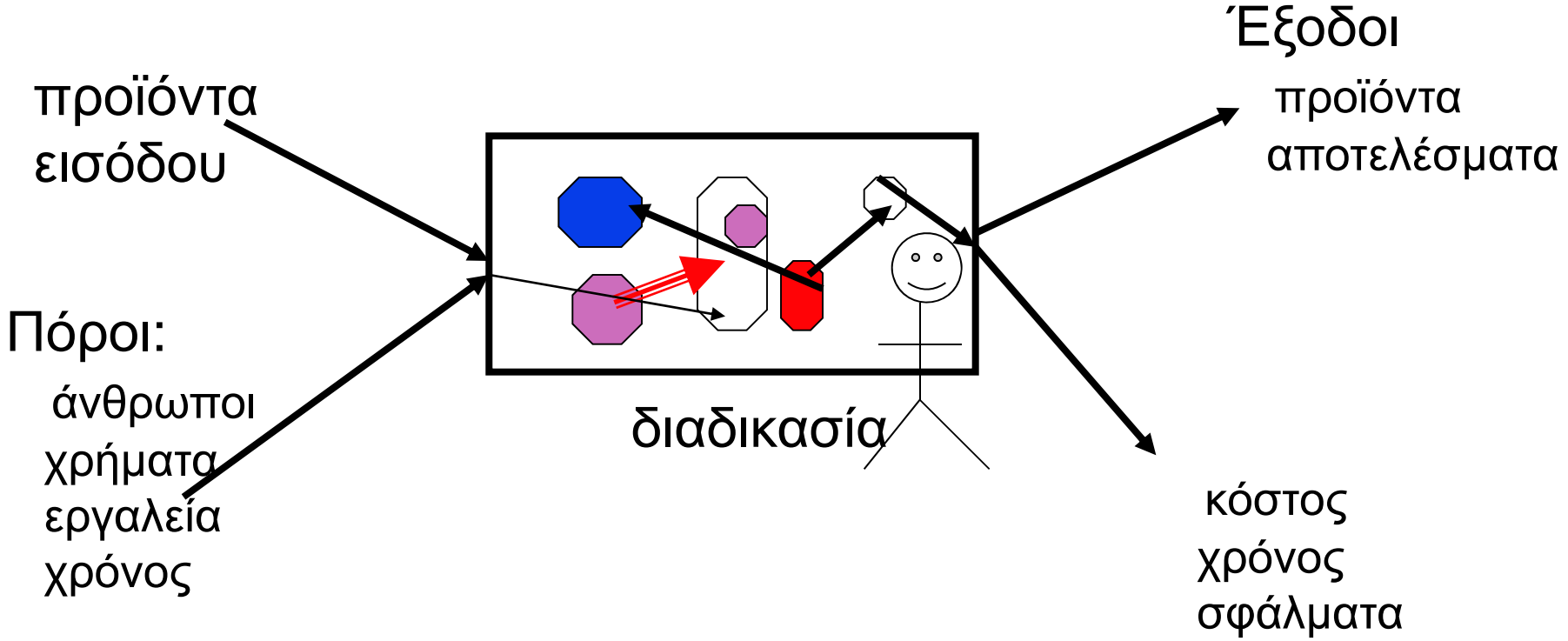
- **Μακροσκοπική**
  - **Τι κάνει/πώς συμπεριφέρεται**
- **Μικροσκοπική**
  - **Πώς κάτι επηρεάζει τη συμπεριφορά**

Η μία σημαντική για την άλλη

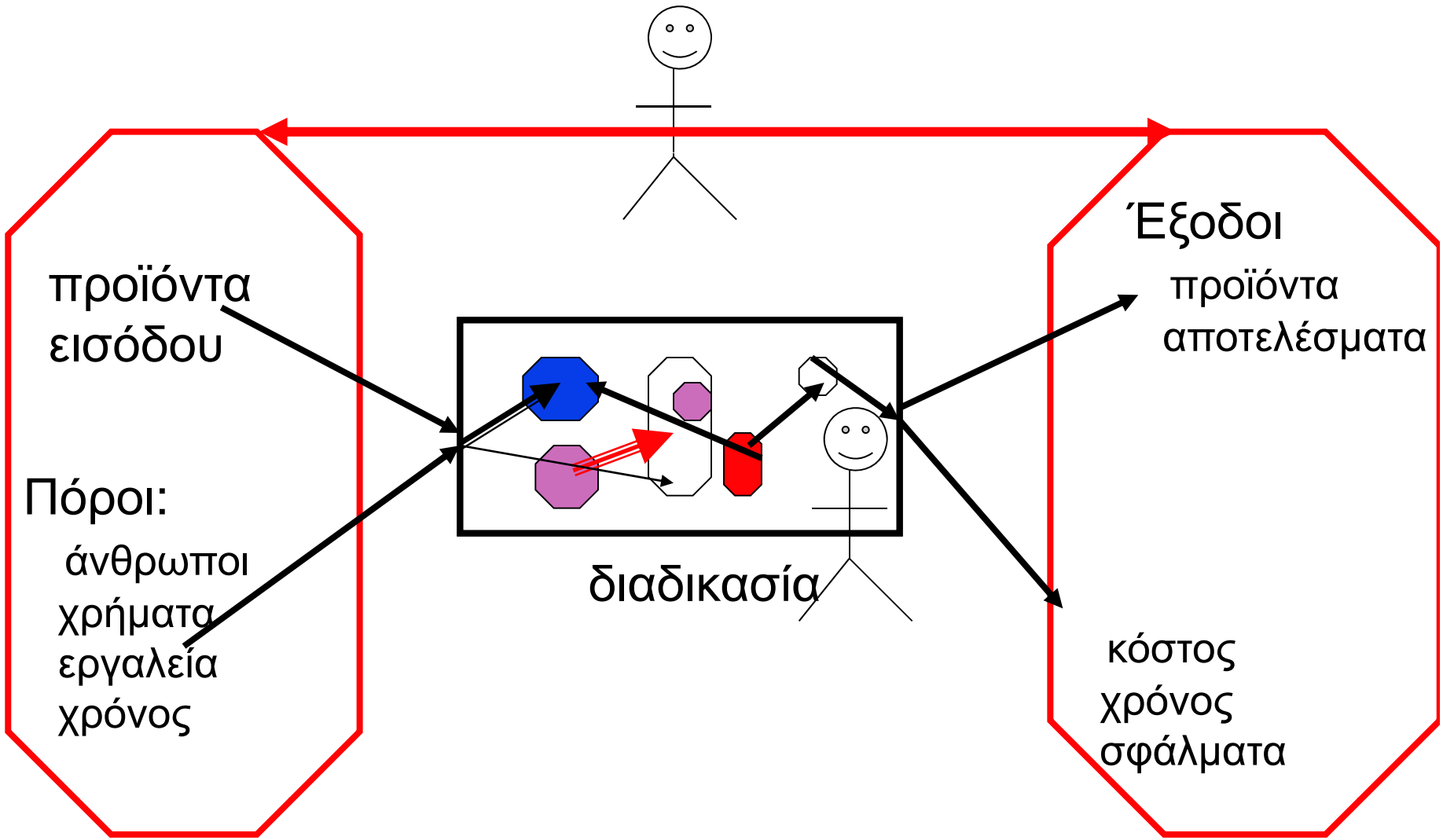
Μακροσκοπική θεώρηση



Μικροσκοπική έμφαση: πιο λεπτομερής συνάρτηση



Η καθεμιά ωφελεί την άλλη



Μπορείτε να φανταστείτε (ακραία) παραδείγματα;

- «Οι προγραμματιστής πρέπει να πίνουν πορτοκαλάδα κάθε πρωί για τις βιταμίνες τους»
  - μακροσκοπική ή μικροσκοπική διαδικασία;
- «Η διαδικασία είναι καλή όταν στην αρχή της εβδομάδας ο προγραμματιστής ξέρει σε ποιο σημείο θα είναι την Πέμπτη στις 4μμ.»
  - μακροσκοπική ή μικροσκοπική θεώρηση;
- Ποια στοιχεία διαδικασίας έχει η δική σας εργασία;
  - μακροσκοπική ή μικροσκοπική

Αναλογία με άλλους τομείς

- Οικονομικά
- Φυσική
  - » Θερμοδυναμική
  - » Ηλεκτρισμός
- Ιατρική/βιολογία

- Τι μαθαίνουμε από τις αναλογίες;
  - η μακροσκοπική θεώρηση έρχεται πρώτη
  - όσο καταλαβαίνουμε κάτι καλύτερα, μικροσκοπικές θεωρήσεις αναπτύσονται
    - » καλύτερος έλεγχος

Συγκεκριμένες Μακροσκοπικές Διαδικασίες

- Capability Maturity Model (CMM)
  - HCMM
  - SCMM
  - κλπ.
- ISO 9000
- Six Sigma
  - 3.4 DPMO (defects per million opportunities)
- TicKIT
- CMMI (Integrated CMM)

Το CMM

- Ορίζει ~30 περιοχές-κλειδί διαδικασίας (Key Process Areas - KPAs)
- Η κάθε μια με πρακτικές-κλειδί
- Αυτές είναι οι διαστάσεις εισόδου-εξόδου που μετρούνται
- Παραδείγματα από KPA
  - Ορισμός διαδικασίας
  - Σχεδιασμός διαδικασίας
  - Χειρισμός απαιτήσεων
  - Project Integration
  - Μέτρηση και ανάλυση

Μέτρηση των KPA

- Μέσω ερωτηματολογίων
  - ~100 ερωτήσεις
  - συμπληρωματικές συνεντεύξεις
- Η τελική μέτρηση γίνεται με την κρίση του εκτιμητή
- Διάστασεις εξόδου
  - Προβλεψιμότητα
  - Επαναληψιμότητα

Βαθμολογίες CMM

- Πέντε επίπεδα ωριμότητας διαδικασίας
- Το μοντέλο προσπαθεί να εκτιμήσει την προβλεψιμότητα μιας διαδικασίας
- Υποτίθεται ότι υψηλότερες βαθμολογίες είναι ενδείξεις ανταγωνιστικού πλεονεκτήματος της ομάδας/οργανισμού
  - σε τι λογισμικό ταιριάζει αυτή η εκτίμηση;

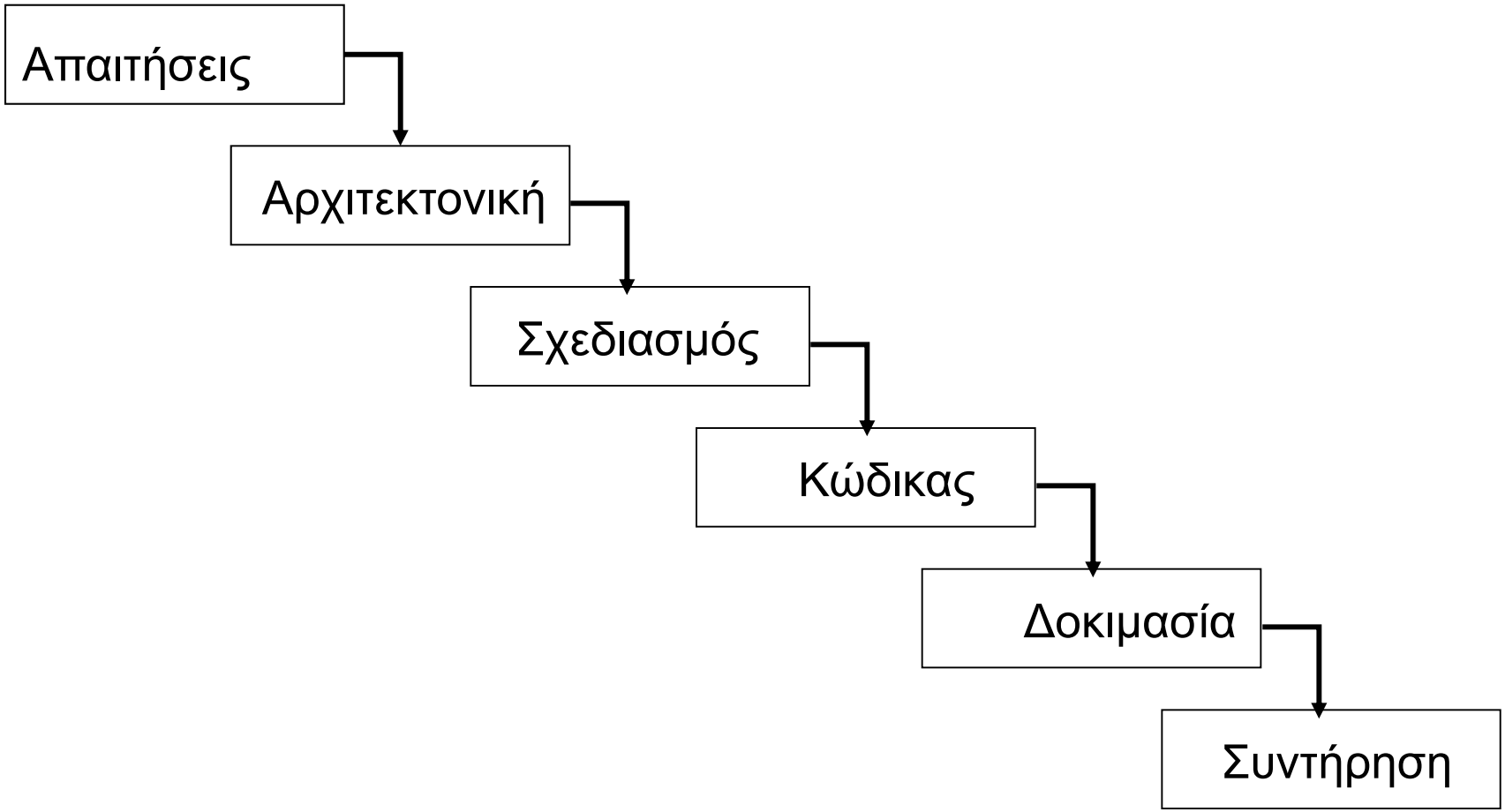
Επίπεδο 1: Αρχικό
Επίπεδο 2: Επαναλήψιμο
Επίπεδο 3: Ορισμένο
Επίπεδο 4: Διαχειριζόμενο
Επίπεδο 5: Βελτιστοποιούμενο

Μικροσκοπικές θεωρήσεις

- Το CMM δεν λέει τίποτα για το πώς να αναπτυξουμε καλύτερες διαδικασίες

- Οι μικροσκοπικές θεωρήσεις διαδικασιών προσπαθούν να αναπτύξουν μοντέλα για την ίδια τη διαδικασία
  - και να τη μελετήσουν όπως μελετάμε το ίδιο το λογισμικό
  - π.χ. η διαδικασία αναπαριστάται από διαγράμματα, η ψευδο-προγράμματα

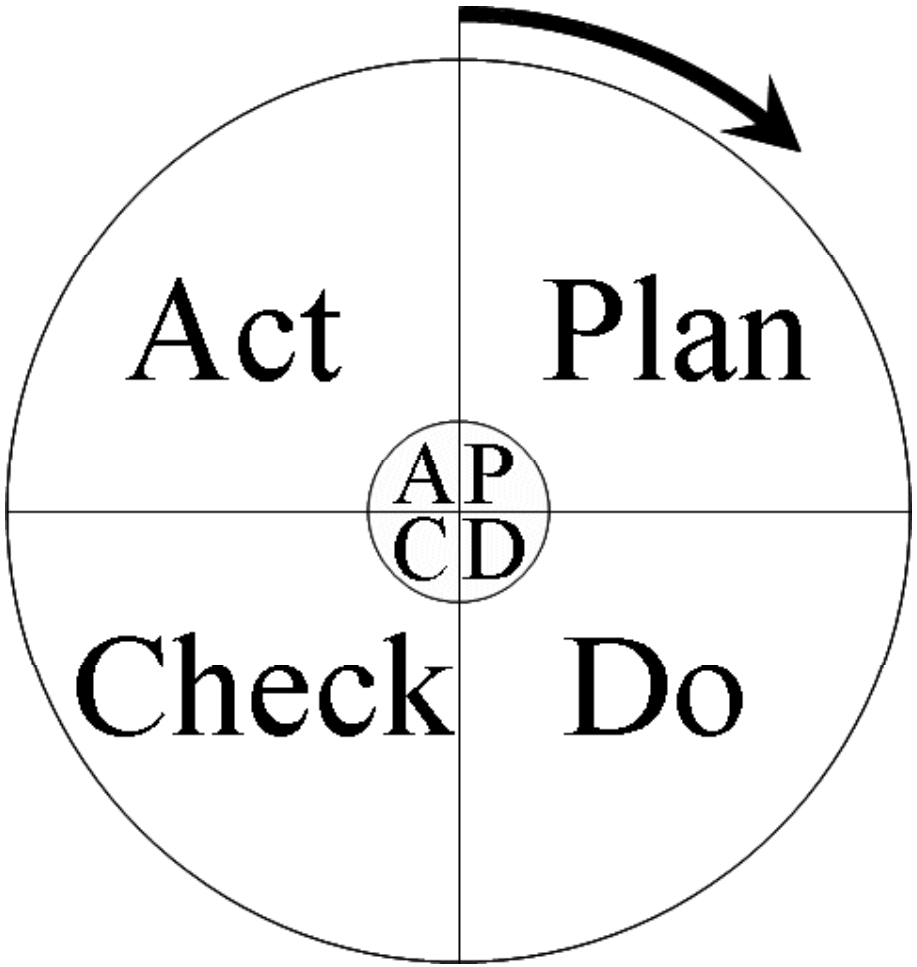
Η διαδικασία «καταράκτη»



Μη ρεαλιστική

- **Υποθέτει ότι η ανάπτυξη λογισμικού δεν πάει ποτέ πίσω**
  - **εντελώς εσφαλμένη υπόθεση**

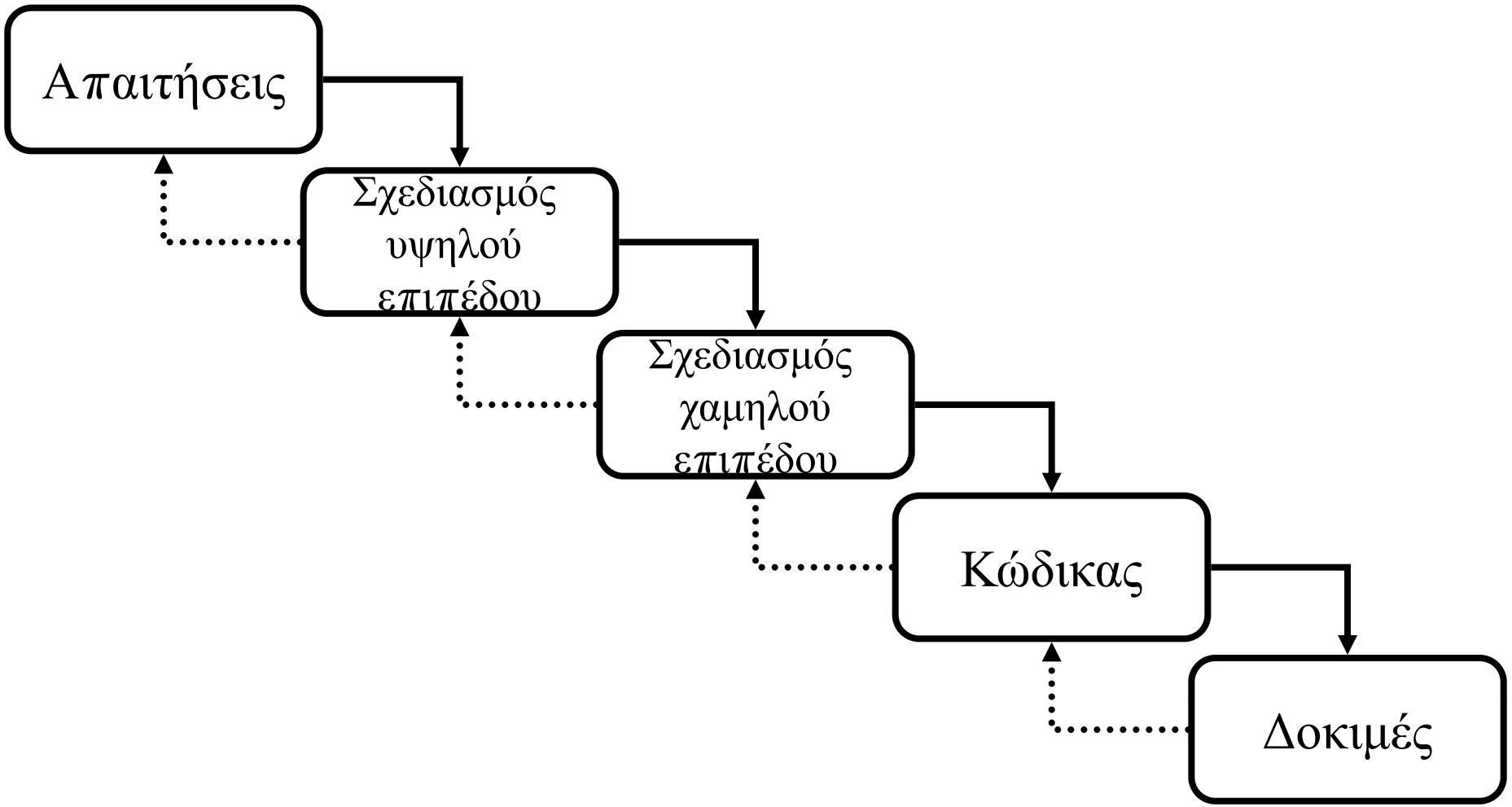
Ο κύκλος Shewhart/Deming
Κάθε επανάληψη βελτιώνει την ποιότητα



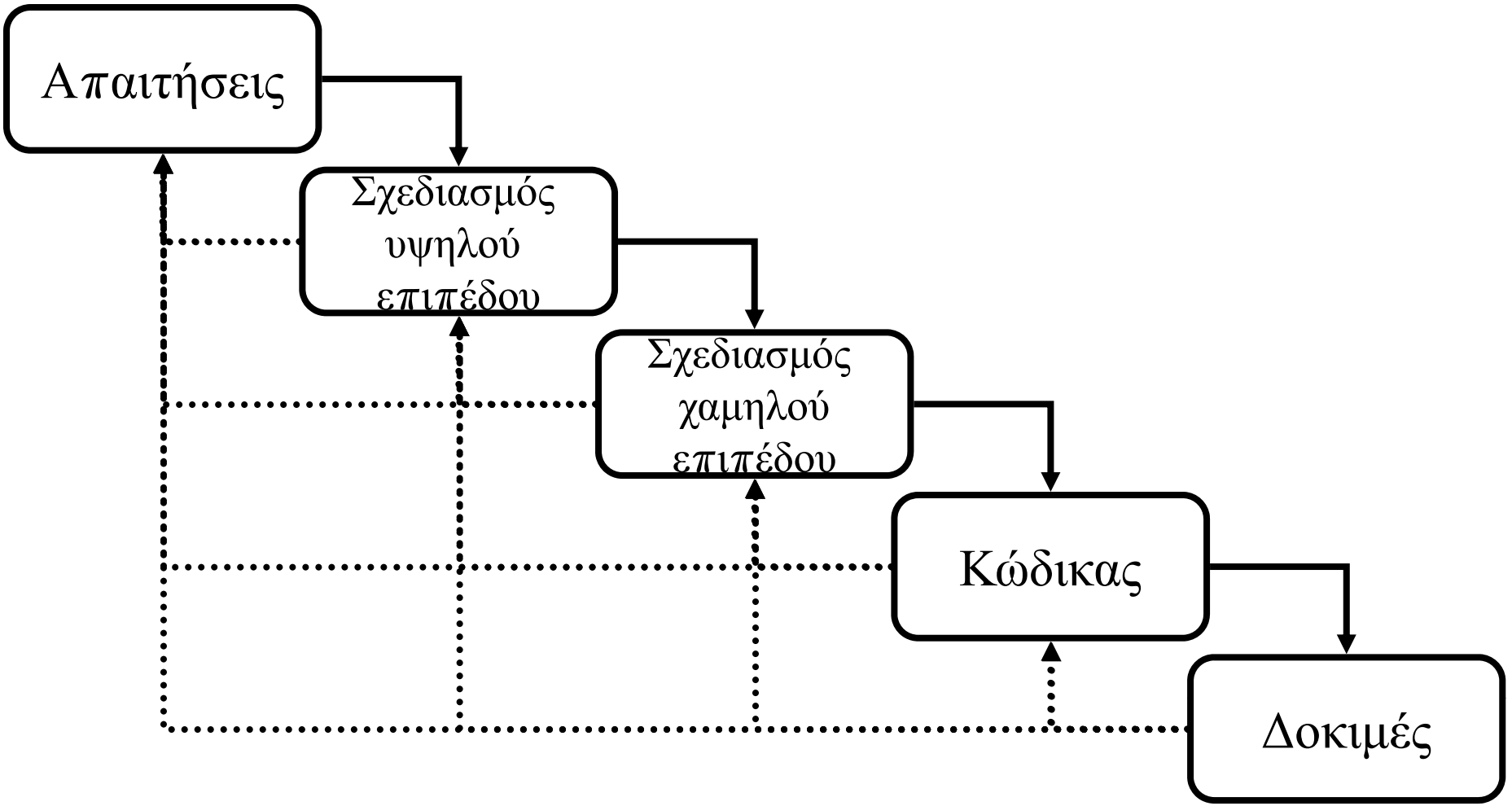
W. Edwards Deming

- Ο πατέρας του μοντέρνου σχεδιασμού ποιότητας στη βιομηχανία
  - Bell Labs τη δεκαετία του 1940
  - μεγάλη αναγνώριση πρώτα στην Ιαπωνία, αργότερα παγκοσμίως
- Έκανε δημοφιλές το "Plan-Do-Check-Act"
- Deming, W. Edwards (1986). *Out of the Crisis*. MIT Center for Advanced Engineering Study. ISBN 0-911379-01-0.
- Αποδίδει το PDCA στον Walter Shewhart
  - Κάτι σαν την «επιστημονική μέθοδο»; (Francis Bacon το 17ο αιώνα)

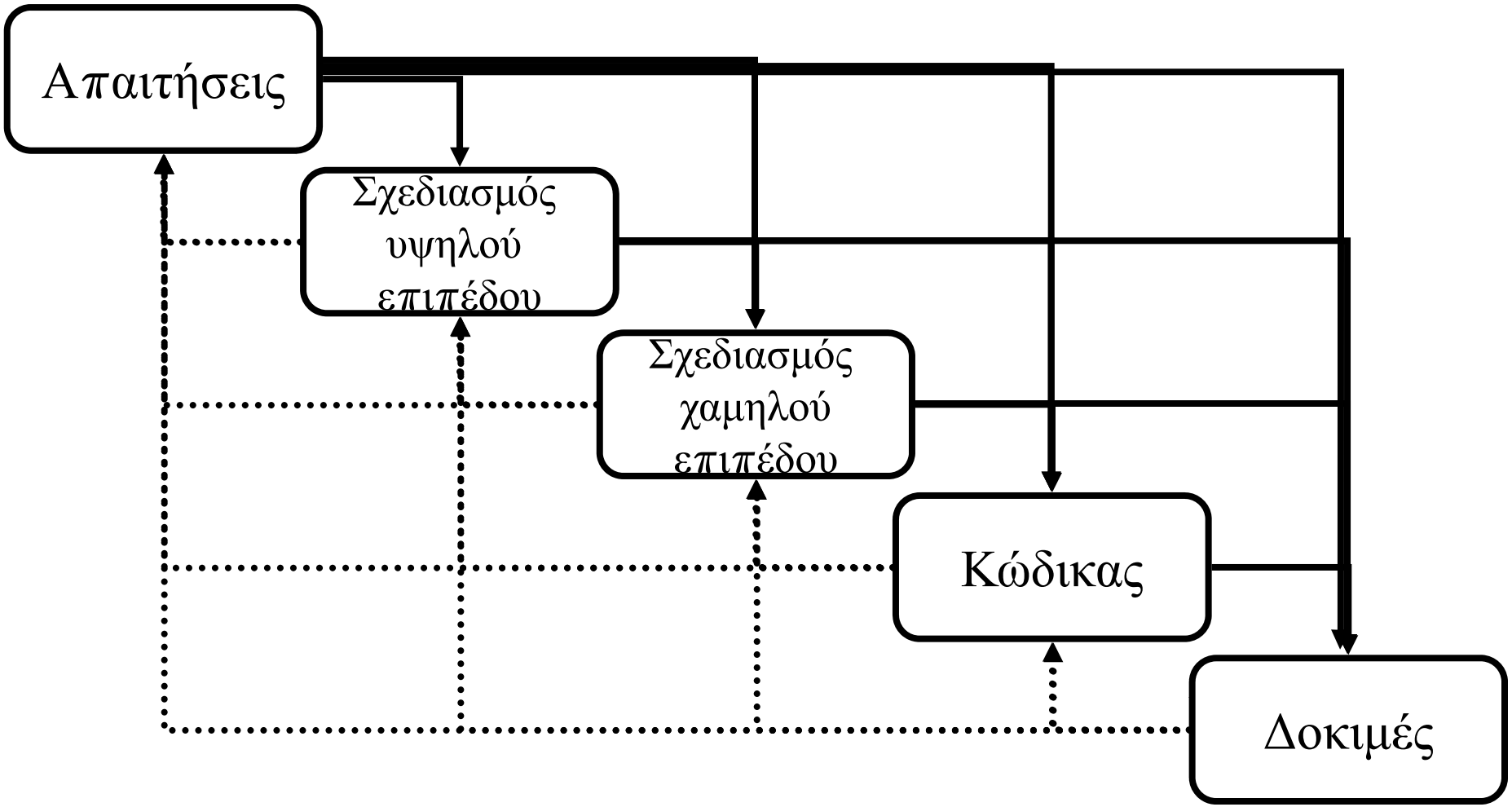
Καταράκτης με κύκλους Shewhart



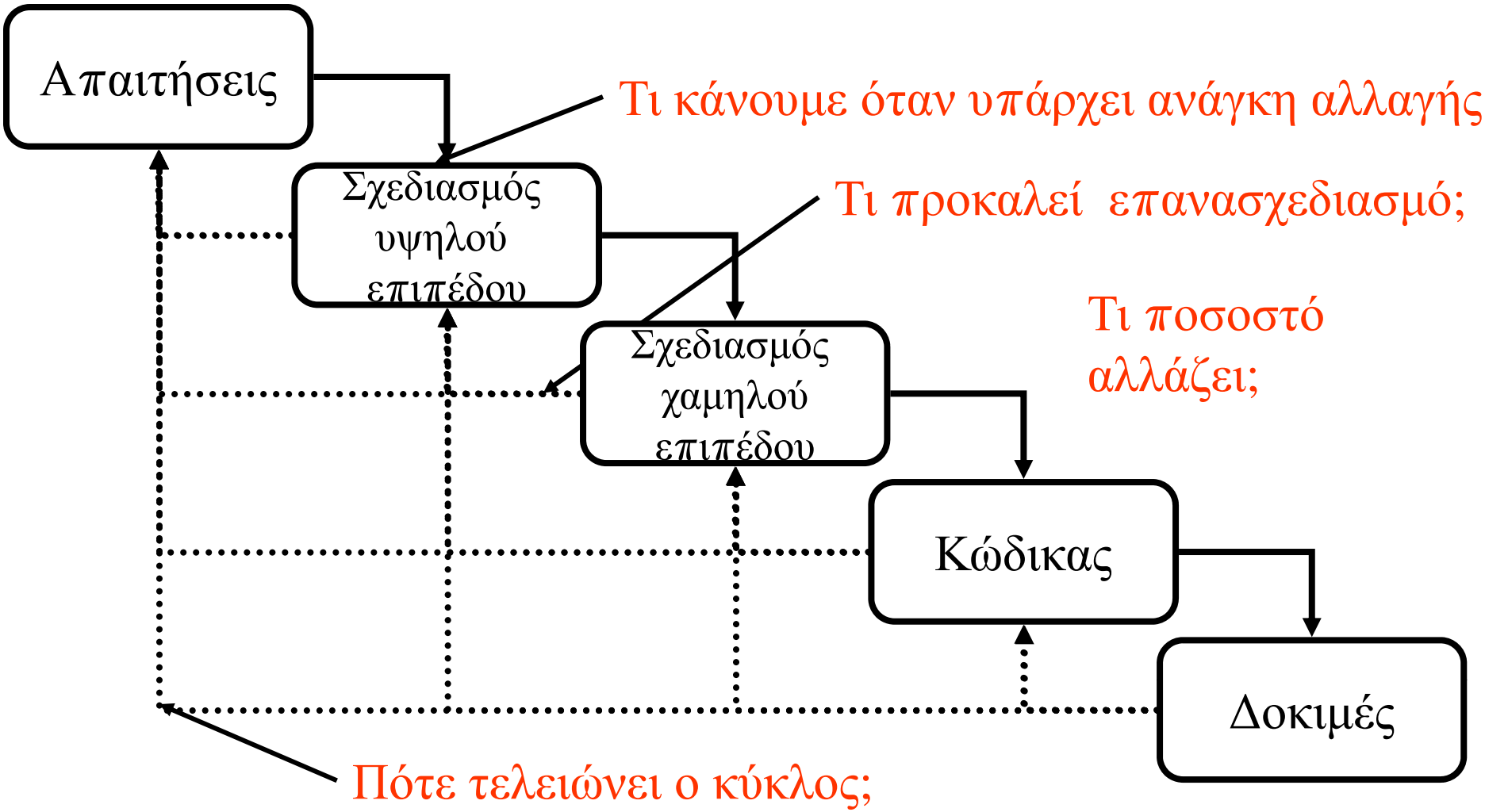
Καταράκτης με πιο πολύπλοκους κύκλους Shewhart



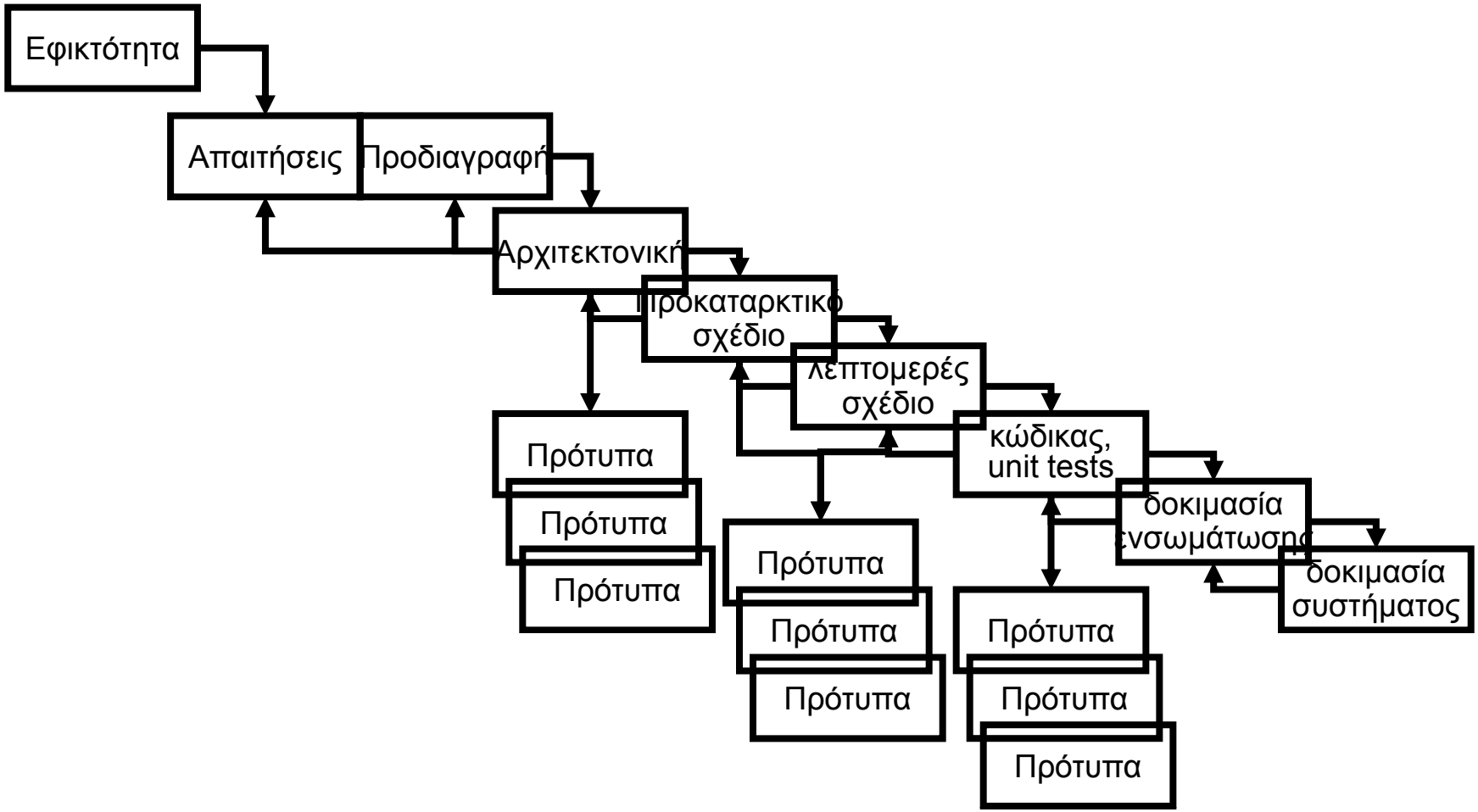
Καταράκτης με πιο πολύπλοκους κύκλους Shewhart και ροή δεδομένων



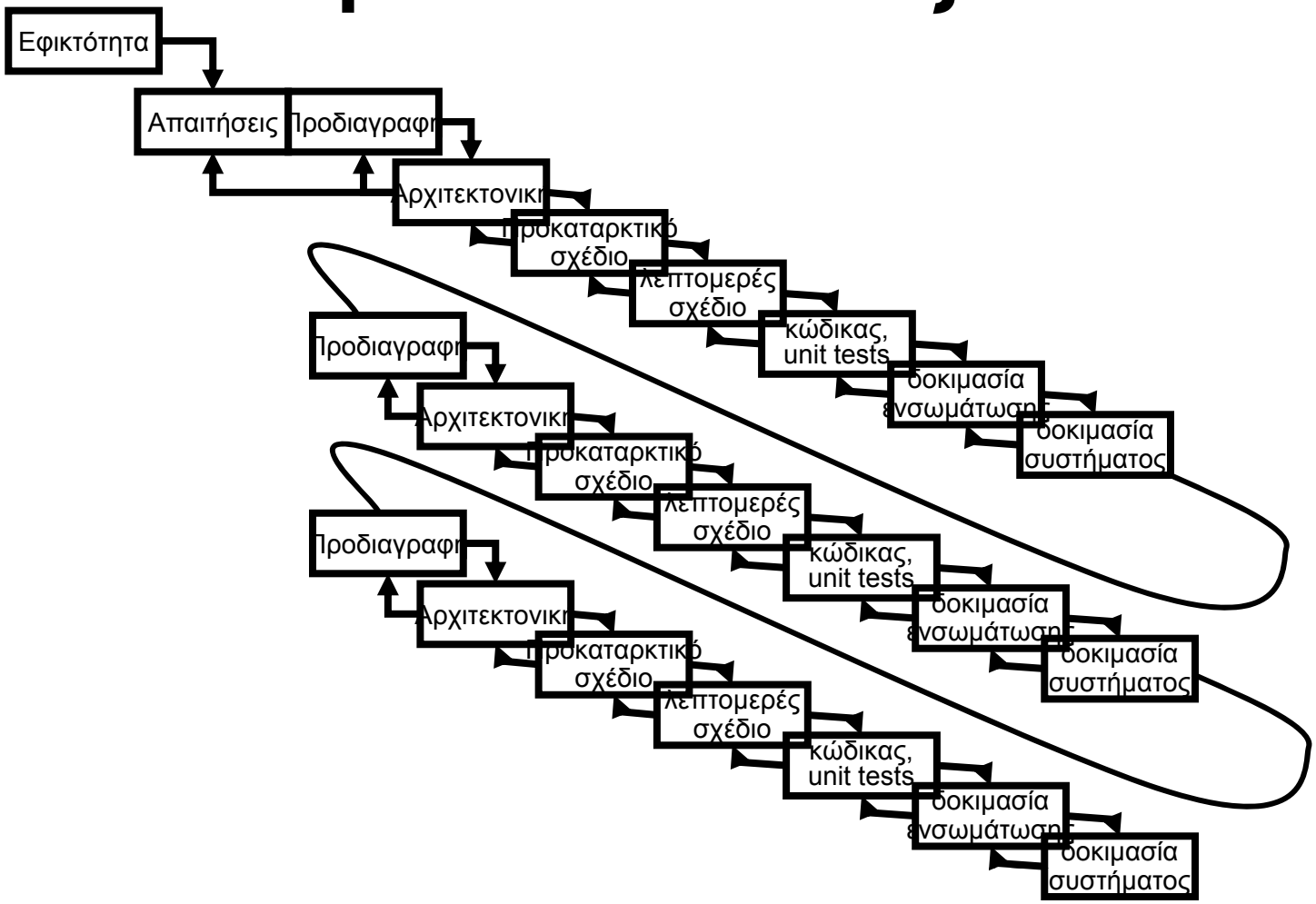
Δεν απαντάει το πώς και πότε γίνεται κάτι



Πρότυπα που πετάγονται



Πρότυπα που εξελίσονται



Extreme Programming (XP)

- «Ακραίος προγραμματισμός»: αντίδραση
  - στην υπερβολική λεπτομέρεια διαδικασίας
  - στην υπερβολική γραφειοκρατία
  - στην έλλειψη έμφασης στην τεχνική δουλειά
    - » ιδιαίτερα τη συγγραφή κώδικα
- Έμφαση στη γρήγορη δημιουργία κώδικα που τρέχει
  - γρήγοροι κύκλοι
  - φιλοσοφία «αν είναι κάτι να πάει στραβά, ας πάει στραβά νωρίς»

Κάποιες προσεγγίσεις XP

- **Δοκιμασίες-πρώτα (test-first programming)**
- **προγραμματισμός σε ζεύγη (pair programming)**
- **Scrum**

Δοκιμασίες πρώτα

- Δοκίμασε το προϊόν πριν το φτιάξεις (!)
- Τι σημαίνει;
  - δημιούργησε λεπτομερές πλάνο δοκιμασίας
  - φτιάξε ένα σκελετό που τρέχει
  - δοκίμασέ το (π.χ. δώστο σε χρήστες)
  - κάνε αλλαγές αμέσως
- Προφανώς μεγάλο τμήμα του προϊόντος απλά εξομοιώνεται
- Πλεονεκτήματα;

Πιο ρεαλιστικό (όχι τόσο XP): Test-Driven Development (TDD)

Μια μέθοδος για να γράφεις κώδικα *είτε για να διορθωθούν σφάλματα είτε για να προστεθεί λειτουργικότητα*

1. **Πρόσθεσε δοκιμασία**
2. **Δοκίμασε ότι πράγματι αποτυγχάνει**
3. **Γράψε κώδικα**
4. **Δες ότι η δοκιμασία περνάει**
5. **Αναδιαμόρφωσε (refactor)**
   - **αν χρειάζεται, χωρίς αλλαγή συμπεριφοράς**

(τυπική σειρά: “1234αποτυχία34επιτυχία“ ή "1234αποτυχία34επιτυχία, είδα κάτι άλλο που θέλει φτιάξιμο 234επιτυχία")

Προγραμματισμός σε ζεύγη

- Ο κώδικας γράφεται σε ομάδες των δύο
- Ένα άτομο “οδηγάει” (κρατάει το πληκτρολόγιο)
- Το άλλο είναι “πλοηγός” (καθοδηγεί, κάνει κριτική, αντιδρά)
  - αλλά δεν γράφει κώδικα!
- Πειράματα δείχνουν ότι μπορεί να είναι πιο παραγωγικό από το να έχεις δύο άτομα που δουλεύουν ανεξάρτητα σε διαφορετικό κώδικα
- Δουλεύει και για άλλα προϊόντα λογισμικού (σχέδια, πλάνα δοκιμασίας, προδιαγραφές)

Scrum

- Η ανάπτυξη διαιρείται σε δεκαπενθήμερες «διαδρομές μικρών αποστάσεων» (“sprints”)
- Οι διαδρομές ξεκινάνε με συνάντηση για καθορισμό στόχων
  - «ποιοι είναι οι κύριοι κίνδυνοι;»
- Ημερίσιες σύντομες συναντήσεις για ενημέρωση
  - όλη η ομάδα παρούσα και δίνει αναφορά
    - » (όρθιοι; ☺)

Η καλύτερη διαδικασία εξαρτάται από τις περιστάσεις

- Τα μέλη της ομάδας, το μέγεθος, την περιοχή, το χρονισμό, κτλ.
- Οι προσεγγίσεις XP είναι μάλλον καλύτερες για
  - μικρές ομάδες
  - μικρότερα προϊόντα
  - προϊόντα που δεν είναι ανάγκη να δουλεύουν τέλεια
  - προϊόντα που ο χρόνος παράδοσης είναι εξαιρετικά σημαντικός

Ας ξαναδούμε κάποια πράγματα που είπαν προγραμματιστές

- *Αν ήμουν υπεύθυνος ομάδας θα έδινα έμφαση στη γρήγορη ανάπτυξη πρότυπης υλοποίησης (rapid prototyping). Όσο πιο λίγο περιπετειώδες είναι ένα έργο τόσο λιγότερο χρειάζεσαι πρότυπα υλοποίησης, αλλά αν δεν υπάρχουν **πολύ πλήρεις προδιαγραφές**, θα έλεγα ότι μια πρότυπη υλοποίηση αξίζει τον κόπο και με το παραπάνω. Μπορεί κανείς να χρησιμοποιήσει το πρότυπο για να φτιάξει ένα αρχικό σύνολο δοκιμών, αν και είναι μάλλον καλύτερο να κρατηθεί το πρότυπο μακριά από τους προγραμματιστές όταν αρχίσουν την πραγματική υλοποίηση, αλλιώς απλά θα εξομοιώσουν το πρότυπο.*
- *Το "Scrum" είναι δημοφιλής μέθοδος τελευταία. Έχω πάρει μόνο μια ιδέα αλλά έχει σύντομους στόχους (milestones), καλές προτεραιότητες, και πολύ άμεσο πάρε-δώσε μεταξύ των μελών της ομάδας. Όλα αυτά ακούγονται πολύ καλά για μικρές ομάδες.*

Περί μάκρο-διαδικασιών

- *Έχω παρατηρήσει ότι όταν κάποιος έχει να ακολουθήσει μια διαδικασία (π.χ. συμπλήρωσε μια φόρμα για το «μοντέλο απειλών ασφαλείας») το κάνει πολύ ευχαρίστως γιατί σημαίνει ότι σημειώνει συγκεκριμένη πρόοδο για εκείνη την ώρα της ημέρας. Είναι πολύ εύκολο να γεμίσεις τη μέρα ενός εργαζομένου με τέτοια πράγματα και θα έχουν υψηλή προτεραιότητα γιατί είναι πολύ πιο εύκολο για το manager να πει «δεν συμπλήρωσες τη φόρμα Χ» παρά να πει «δεν γράφεις καλό κώδικα» ή «η πρόοδος σου είναι αργή».*

Διαδικασίες και Μέγεθος Έργου

- *Μεγάλες ομάδες που δουλεύουν σε μεγάλες βάσεις κώδικα με περισσότερη πολυπλοκότητα πρέπει να έχουν πολλή «διαδικασία». Οι καλές μεγάλες ομάδες θα διαλέξουν το έλαχιστο διαδικασίας που θα δώσει το μέγιστο πλεονέκτημα. Όμως εν τέλει, αν κάποιος δεν πολυθέλει διαδικασίες θα είναι πιο ευτυχισμένος σε μικρότερες ομάδες/προϊόντα.*

Διαδικασία και Σχεδιασμός

- *Ένας φίλος μου είναι σε μια άλλη ομάδα στην [εταιρία] και έχουν «κρίση ποιότητας». Στα τελευταία τους milestones είχαν βαθμό οπισθοδρόμησης (regression rate) 30% ανά checkin (δεν έχω ιδέα τι στατιστικά θεωρούνται φυσιολογικά γενικά αλλά 30% δεν μου φαίνεται και υπερβολικά μεγάλο – ίσως 15% είναι πιο λογικό;) κι έτσι η «λύση» είναι ότι ένας από τους αρχιτέκτονες (που ο φίλος μου τον θεωρεί πανάχρηστο και ηλίθιο) προσπαθεί να επιβάλει κάποια μέτρα ποιότητας (π.χ. κυκλωματική πολυπλοκότητα, ποσοστό σχολίων ανά γραμμή κώδικα, κτλ.) που θα επιβάλονται αυτόματα στο checkin για να βελτιώσουν τον κώδικα. Ο φίλος μου μου έστειλε το κείμενο αυτού του τύπου και βασικά έλεγε «για να διορθώσουμε τα προβλήματά μας θα κάνουμε αυτό» με κάποια μέτρα ποιότητας που τα έβγαλε απ' το κεφάλι του χωρίς καμμία λογική σύνδεση με το πρόβλημα και χωρίς κανένα πλάνο για να εκτιμήσει αν η νέα διαδικασία δουλεύει.*

Διαδικασία και Δοκιμασίες

- *Μια διαδικασία που πραγματικά μ'αρέσει είναι η ανάπτυξη που καθοδηγείται από δοκιμασίες (test-driven development). Έγραψα στο blog μου γι'αυτό πριν λίγο καιρό [...] Λέω επίσης για την «κάλυψη κώδικα» σαν ένα χρήσιμο μέτρο ποιότητας. Είναι το μόνο μέτρο που ξέρω που το χρησιμοποιούν σχεδόν όλοι. Η ιδέα είναι ότι αν έχεις λιγότερο από ~80% κάλυψη τεμαχίων (block coverage) από τα test σου τότε κάτι πάει στραβά.*

Διαδικασία και Δοκιμασίες

- *Η καλύτερη προσέγγιση στην ανάπτυξη λογισμικού είναι να βρεις ένα τρόπο να κάνεις «τα πάντα δυο φορές», κάτι σαν το διπλογραφικό σύστημα στη λογιστική. Σε κάποιες περιπτώσεις βασίζεσαι στο σύστημα τύπων. Σε άλλες γράφεις asserts. Σε άλλες μοναδιαίες δοκιμασίες. Θα έλεγα ότι αυτό είναι το πιο σημαντικό: υπάρχει κάποιος τρόπος που το μηχάνημα να ελέγξει το κάθε στοιχείο ενός προγράμματος έναντι κάποιου άλλου στοιχείου; Έλεγξε όσο γίνεται, όσο πιο νωρίς γίνεται, χρησιμοποιώντας compile-time asserts και πολύπλοκους τύπους που θα μεταγλωττιστούν σε σχεδόν τίποτα. (Βρήκα ένα σωρό bugs πριν κανά-δυο μήνες όταν άλλαξα ένα typedef που χρησιμοποιούταν με ελαφρά διαφορετικούς τρόπους σε ένα class template με 4 ασύμβατα instantiations που δεν μπορούσες να χρησιμοποιήσεις το ένα αντί για το άλλο χωρίς μετατροπή.) Μια καλή ιδέα είναι ένα ξεχωριστό σύστημα κατασκευής που τρέχει αυτόματες δοκιμασίες πολύ αργά αλλά κάνει πολλούς ελέγχους συνέπειας σε όλες τις δομές δεδομένων.*

Διαδικασία και Δοκιμασίες

- [Από τον ίδιο που είπε 3dev + 3test + 1PM = 2dev]
  Ένα πρόβλημα που έχουμε με δοκιμαστές που δεν κάνουν τίποτε άλλο είναι ότι δεν υπάρχει τρόπος ελέγχου τους. Λένε απλά «αυτό δοκιμάστηκε» και όλοι (ή μάλλον ο manager) τους πιστεύουν. Πρότεινα δύο πιθανά επίπεδα ελέγχου. Το πρώτο είναι να επιτρέψουμε στους προγραμματιστές να βάλουν σε συγκεκριμένα σημεία του κώδικα κάποιο macro ή κάτι τέτοιο που να σημαίνει «αυτή η περίπτωση πρέπει να καλυφθεί». Οι δοκιμασίες θα πρέπει να καλύψουν όλες αυτές τις περιπτώσεις. Οι δοκιμαστές μας στηρίζονται σε μέτρα κάλυψης και εξ'αιτίας διάφορων περιπτώσεων για χειρισμό λαθών ένα 70% θεωρείται αποδεκτό. Κανείς δεν ξέρει αν σημαντικές περιπτώσεις μένουν χωρίς έλεγχο. Το ακόμα υψηλότερο επίπεδο ελέγχου θα ήταν να επιτρέψουμε στους προγραμματιστές να προσθέσουν macros που θα εισάγουν bugs επίτηδες σε κάποιο ειδικό build.
- [άλλος προγραμματιστής, χωρίς να απαντάει στον προηγούμενο]
  Κοίτα, μιλάω από εμπειρία. Οι δοκιμαστές μας βρίσκουν ένα σωρό bugs.