HY352 : ΤΕΧΝΟΛΟΓΙΑ ΛΟΓΙΣΜΙΚΟΥ

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΡΗΤΗΣ,
ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ,
ΤΜΗΜΑ ΕΠΙΣΤΗΜΗΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΔΑΣΚΩΝ
Αντώνιος Σαββίδης

CSD

Γενικές πληροφορίες (1/2)

- Τρόπος βαθμολογίας:
  - 70% τελική εξέταση >= 5
  - 25 - 30% project – αρκετή δουλειά >= 5
- Σχολαστικός έλεγχος αντιγραφών στις ασκήσεις
  - Αντιγραφή = μηδενισμός.
- Σημειώσεις είναι το περιεχόμενο των διαλέξεων στο site του μαθήματος, την επόμενη μέρα της κάθε διάλεξης.
  - Θα χρειαστεί να κρατάτε σημειώσεις για επεξηγήσεις οι οποίες δεν θα υπάρχουν στις διαφάνειες

CSD

Γενικές πληροφορίες (2/2)

- Η γλώσσα προγραμματισμού του μαθήματος είναι η C++
  - International Standard ISO/IEC 14882:2014(E) Programming Language C++
  - Γνωστή πλέον ως το standard **C++14** (18/8/2014)
- Έχουν οργανωθεί φροντιστήρια σε C++ (κάθε εβδομάδα, το προγραμματισμένο φροντιστήριο)
  - Συνίσταται η συμμετοχή σας σε *όλα* τα φροντιστήρια
- Το βιβλίο είναι το «*Η γλώσσα προγραμματισμού C++*» του Bjarne Stroustroup (σχεδιαστή της C++), Ελληνική μετάφραση

CSD

Εισαγωγή

Γενικές αρχές

- Καλή μελέτη του **προβλήματος** και των σχετικών παραμέτρων
  - **Ξέρουμε πλήρως τι πρόβλημα λύνει το λογισμικό που πρόκειται να κατασκευάσουμε**
- Καλή γνώση της διαθέσιμης **τεχνολογίας** ανάπτυξης και των προδιαγραφών λειτουργίας
  - **Είμαστε καλοί γνώστες των εργαλείων κατασκευής, και γνωρίζουμε ακριβώς πως πρέπει να συμπεριφέρεται το λογισμικό σε όλες τις συνθήκες**
- Καλή γνώση ύπαρξης **λύσεων** σε παρόμοια προβλήματα
  - **Ξέρουμε την μεθοδολογία που έχουν ακολουθήσει άλλοι στην επίλυση του ίδιου προβλήματος**

Κύριοι στόχοι

- Ελαχιστοποίηση λαθών – **minimize bugs**
- Μείωση όγκου υλοποίησης – **minimize code size**
- Εύκολη τροποποίηση – **minimize dependencies**
- Διατήρηση του συστήματος – **maximize lifetime**
- Επαναχρησιμοποίηση – **maximize reuse**
- Επεκτασιμότητα σχεδίασης – **maximize extensibility**

➔ **Πως κατασκευάζουμε καλύτερο software στο μικρότερο χρόνο και στο μικρότερο κόστος**

Τι θα μάθετε

- Διαδικασία παραγωγής λογισμικού ✓
- Αρχιτεκτονική σχεδίαση ✓✓
- Σχεδίασης υλοποίησης ✓✓✓
- Δομημένος προγραμματισμός ✓
- Στοιχεία αντικειμενοστραφούς προγραμματισμού ✓✓✓✓
- Σχεδιαστικά πρότυπα ✓✓✓✓✓
- Οδηγίες καλού προγραμματισμού ✓
- Τεχνικές εντοπισμού και επιδιόρθωσης λαθών ✓
- Ακραίο προγραμματισμό ✓

Γενικά χαρακτηριστικά

- Ή διαδικασία ανάπτυξης κατευθύνεται από τις ανάγκες που καλύπτει το σύστημα - ***requirements driven***
  - είναι συνηθισμένο να βιαζόμαστε να σχεδιάσουμε πριν αποφασίσουμε το σύνολο των δυνατοτήτων του συστήματος
- Σαφής διαχωρισμός της ***αρχιτεκτονικής σχεδίασης***
  - που είναι η πρώτη σχεδιαστική διαδικασία και αποτυπώνει τα λειτουργικά τμήματα και τις αλληλεπιδράσεις ενός συστήματος
- από τη ***σχεδίαση της υλοποίησης (δηλ. του κώδικα)***
  - που έπεται και ορίζει πως θα υλοποιηθεί η αρχιτεκτονική, με τον καλύτερο τρόπο, σε κώδικα σε συγκεκριμένη γλώσσα

➢ **Μεγάλα λάθη γίνονται σε όλες τις διαδικασίες με πιο συνηθισμένο να ξεχνάμε βασικές απαιτήσεις**

Επιγράμματα (Alan Perlis, 1982)

- Every program is a part of some other program and rarely fits
- It is easier to write an incorrect program than understand a correct one
- Everything should be built top-down, except the first time
- Optimization hinders evolution
- Simplicity does not precede complexity, but follows it
- It is easier to change the specification to fit the program than vice versa
- In computing, the mean time to failure keeps getting shorter
- Whenever two programmers meet to criticize their programs, both are silent

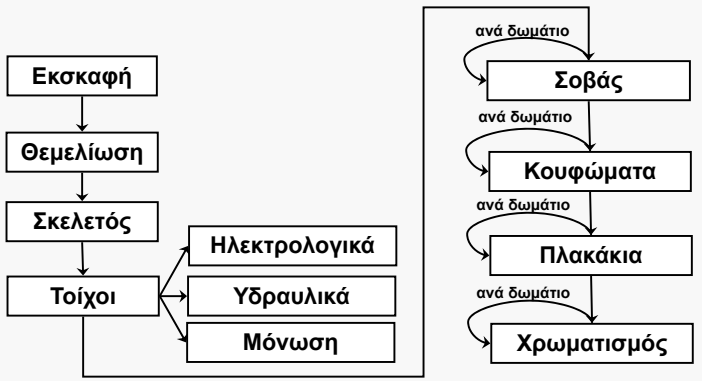
Ένα σχετικό παράδειγμα (1/3)

- ***Κατασκευή κτιρίων***
  - Γενικό αρχιτεκτονικό σχέδιο (διαρρύθμιση, χώροι).
  - Λεπτομερές τεχνικό σχέδιο (μηχανικό, στατικό).
  - Πρόβλεψη υλικών, κόστους, χρόνου, προσωπικού, και μηχανημάτων.
  - Καλά ορισμένη και τυποποιημένη διαδικασία κατασκευής
    - Θεμέλια
    - Κολώνες – σκελετός
    - Δίκτυα (ύδρευση, αποχέτευση, ηλεκτρικό, τηλεπικοινωνίες, εξαερισμός, θέρμανση)
    - Εσωτερικοί / εξωτερικοί τοίχοι.
    - Κουφώματα, μάρμαρα, χρωματισμός.
    - Εξοπλισμός.

Ένα σχετικό παράδειγμα (2/3)

- Γενικά χαρακτηριστικά της διαδικασίας
  - Η αρχιτεκτονική παίζει τον πρώτο και κύριο ρόλο
  - Πολύ δουλειά και υπολογισμοί «επί χάρτου»
  - Αυστηρή ακολουθία βημάτων με οργανωμένο έλεγχο
  - Κατανεμημένες εργασίες, πολλοί συμμετέχοντες
  - Ανεξαρτησία κατασκευής διαφορετικών τμημάτων
  - Όσο προχωρά η κατασκευή, μειώνεται η εμβέλεια τροποποιήσεων (μόνο σε μικρή τοπική κλίμακα)
  - Μεγάλο κόστος και *αυξημένη επικινδυνότητα σχεδιαστικών λαθών*
  - Πρόβλεψη χρόνου και προϋπολογισμός κόστους
  - Μη γραμμική αύξηση κατασκευαστικής πολυπλοκότητας κατά την γραμμική αύξηση μεγέθους του κατασκευάσματος (π.χ. από πολυκατοικία σε ουρανοξύστη)

Ένα σχετικό παράδειγμα (3/3)



Η διάταξη των εργασιών μπορεί να αποτυπωθεί σε έναν γράφο ο οποίος δείχνει πότε μία εργασία μπορεί να αρχίσει και ποιες εργασίες εξαρτώνται από αυτή

CSD

Διαφορές από άλλες επιστήμες

1. Μεγάλη ποσότητα γνώσης με την μεγαλύτερη ταχύτητα μεταβολής, γεγονός που την καθιστά (τη γνώση) γρήγορα απαρχαιωμένη
2. Δεν υπάρχει άδεια άσκησης επαγγέλματος, δηλ. δεν υπάρχει μέθοδος που να χαρακτηρίζει επαγγελματίες προγραμματιστές
3. Δεν καλλιεργείται η ιδέα ότι τα σφάλματα πρέπει να αντιμετωπίζονται ως άκρως επικίνδυνα, όχι απλώς ανεπιθύμητα
4. Πολύ χαμηλό κόστος υλικοτεχνικής υποδομής για την διαδικασία ανάπτυξης και πειραματισμού
5. Η αντιγραφή της λύσης του ίδιου προβλήματος είναι τετριμμένη
6. Άγνωστη η διαδικασία παραγωγής και οι προκλήσεις της στον τελικό αποδέκτη
7. Απρόσιτη η εσωτερική λειτουργία και η εξήγηση αιτιών των λειτουργικών λαθών στον τελικό αποδέκτη

CSD

Σχετικά γνωμικά

- Διαίρει και βασίλευε
- Το δις εξαμαρτείν ουκ ανδρός σοφού
- Καλύτερα πρόβλεπε, παρά θεράπευε
- Μην εφευρίσκετε ξανά τον τροχό
- Μάτια που δεν βλέπονται γρήγορα λησμονιούνται
- Όπου λαλούν πολλά κοκόρια αργεί να ξημερώσει
- Μην αναβάλλεις για αύριο αυτό που μπορεί να κάνεις σήμερα
- Κάλιο αργά παρά ποτέ
- Τα ράσα δεν κάνουν τον παπά

Programming does not start with coding, but ends with coding

CSD

Προβλήματα ανάπτυξης

- IBM survey, 1994
  - 55% of systems cost more than expected
  - 68% overran schedules
  - 88% had to be seriously redesigned
- US bureau of labour
  - For every 6 new systems put into operation, 2 cancelled
  - Probability of cancellation is %50 for biggest systems
  - 3 to 4 systems are considered as 'operating failures'
- Τα περισσότερα οφείλονται στην έλλειψη συστηματικής σχεδίασης και ανάπτυξης με εφαρμογή κατάλληλων τακτικών
- Η κατανόηση της ανάγκης των μεθόδων τεχνολογίας ανάπτυξης έρχεται μόνο από την εμπειρία, αφού πάθετε πρώτα, δύσκολα θα σας πείσει ένα μάθημα

CSD

Ο ρόλος των απαιτήσεων (1/4)

- Απαιτήσεις = *τι θέλουμε να κάνει το σύστημα μας*
- Πάντοτε ξεκινάμε την κατασκευή με τις απαιτήσεις – ***requirements first***
  - Συνήθως αγνοούμε ή δεν αντιλαμβανόμαστε από την αρχή περίπου το 50% των χαρακτηριστικών ενός συστήματος
  - Και για να φτάσει κανείς σε αυτό το 50% πρέπει να κοπιάσει πάρα πολύ
- ***Ένα σύστημα δεν είναι ουσιαστικά ποτέ έτοιμο***
  - Απλά αναγκαζόμαστε να το παραδώσουμε κάποια στιγμή στην παρούσα κατάστασή (περιορισμοί κυρίως χρόνου ή κόστους)
  - Σταματάμε να εξελίσσουμε ένα σύστημα για δύο λόγους: δεν είναι συμφέρουσα η συγκεκριμένη δραστηριότητα (*business*) ή δεν είναι εφικτή η βελτίωση και επέκταση του κώδικα

Ο ρόλος των απαιτήσεων (2/4)

- Κάθε σύστημα έχει από τον ορισμό του ένα σύνολο από ***κυρίαρχα χαρακτηριστικά*** - primary features
- Εάν το σύστημα είναι κάτι εντελώς νέο, τα χαρακτηριστικά αυτά είναι οι προδιαγραφές που αφορούν καινοτομίες λειτουργικότητας
  - ***Τα ξεχάσατε ή τα αγνοήσατε ⇒ δεν έχετε το σύστημα που θέσατε ως αρχικό σας στόχο***
- Εάν το σύστημα ανήκει σε μία υπάρχουσα κατηγορία, τα χαρακτηριστικά αυτά είναι οι επεκτάσεις και βελτιώσεις λειτουργικότητας ως προς τα ανταγωνιστικά συστήματα
  - Εάν δεν τα λάβετε υπόψη από την αρχή μάλλον θα καταλήξετε στη μεθοδολογία ανάπτυξης των παλαιότερων αντίστοιχων συστημάτων
  - Αυτό μπορεί να σημαίνει ότι θα είναι πολύ δύσκολο να τα ενσωματώσετε στον κώδικα αργότερα
- Τα κυρίαρχα χαρακτηριστικά ***επηρεάζουν αρκετά τον τρόπο ανάπτυξης και πρέπει να προσδιορίζονται πολύ νωρίς***

Ο ρόλος των απαιτήσεων (3/4)

- *Παραδείγματα*
  - *Game*: *οπτική μορφή*, *γεωμετρία κόσμου*, *κανόνες φυσικής*, *είδος δράσης*, *είδος χαρακτήρων*
  - *Language*: *κατηγορία γλώσσας*, *σύστημα τύπων*, *υποστήριξη αντικειμένων*, *τιμές και μεταβλητές*, *εντολές*, *βιβλιοθήκες*

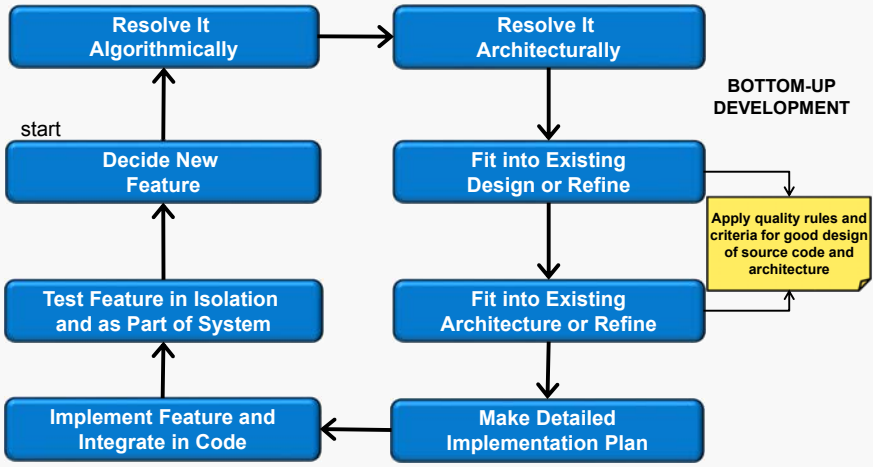
Ο ρόλος των απαιτήσεων (4/4)

- Το πρόβλημα της ανάπτυξης ενός συστήματος είναι απλώς αλγοριθμικό μόνο για μικρά, πολύ μικρά, σε γελοίο βαθμό μικρά συστήματα
- Πλέον διατρέχουμε μία εποχή όπου το μέγεθος του κώδικα πρακτικά εκρήγνυται
- Το πρόβλημα πλέον δεν είναι απλώς κατασκευαστικό ή μηχανικό, δηλαδή φτιάξε το σύστημα με κάποιο τρόπο
  - αλλά και θέμα διαχείρισης, συντήρησης και επέκτασης
  - ο στόχος είναι το λογισμικό να μπορεί να επεκτείνεται και να αναβαθμίζεται για όσο το δυνατόν περισσότερο χρόνο
- Το πρόβλημα είναι σχεδιαστικό, απαιτώντας πολύ καλή γνώση προηγμένων προγραμματιστικών τεχνικών

Top-Down και Bottom-Up (1/3)

- Γενικά υπάρχουν δύο βασικές σχολές σκέψεις ή τακτικές ανάπτυξης
- **Top-down**: πρώτα προσδιορίζεται η αρχιτεκτονική και μόνο στο τέλος υλοποιείται ο κώδικας
  - Πιο ιδανική προσέγγιση ή «*προγραμματιστικός παράδεισος*»
- **Bottom-up**: πρώτα υλοποιούμε τις λειτουργίες σε κώδικα και συνθέτουμε την αρχιτεκτονική λίγο-λίγο
  - Είναι πιο πρακτική προσέγγιση ή «*λίγο βρώμικη*»
- Ο χρυσός κανόνας
  - ***Ένα σύστημα κατασκευάζεται πάντοτε top-down εκτός από την πρώτη φορά που είναι bottom-up***

Top-Down και Bottom-Up (2/3)



Top-Down και Bottom-Up (3/3)

- Γέννηση του συστήματος ως επαναληπτική διαδικασία με τον κώδικα να υλοποιείται πριν καν έχουμε την εικόνα του τι πάμε να φτιάξουμε
- Προφανώς υπάρχει ένας αρχικός αριθμός από features που προσδιορίζονται με ακρίβεια πριν την υλοποίηση
- Ωστόσο υπάρχουν πολλά features τα οποία επιλέγονται κατά τη διάρκεια της υλοποίησης και χωρίς να έχουμε από πριν τεκμηριωμένους τρόπους κατασκευής
- Στην πράξη καινοτομούμε στην κατασκευή του συστήματος, κάτι πολύ συνηθισμένο στο software

Ανάλυση και πρόβλεψη κινδύνων (1/8)

- Η ανάλυση και πρόβλεψη κινδύνων βασίζεται σε συγκεκριμένες διαδικασίες:
  - Ανάλυση απαραίτητης τεχνογνωσίας
    - know how
  - Μελέτη δυνατότητας ανάπτυξης
    - feasibility study
  - Μελέτη συνολικού κόστους
    - cost estimate
  - Παρακολούθηση προόδου
    - progress monitoring

Ανάλυση και πρόβλεψη κινδύνων (2/8)

- ***Know how***
  - Υπάρχει προηγούμενη εμπειρία κατασκευής τέτοιων συστημάτων
  - Υπάρχει καταγεγραμμένη η στρατηγική ανάπτυξης τέτοιων συστημάτων
  - Υπάρχει προσωπικό με γνώση αρχιτεκτονικής, τμημάτων, εργαλείων, αλγορίθμων, υλοποίησης τέτοιων συστημάτων
  - Υπάρχει εμπειρία στην ανάπτυξη συστημάτων αυτού του μεγέθους

CSD

Ανάλυση και πρόβλεψη κινδύνων (3/8)

- ***Feasibility study (1/2)***
  - Υπάρχει δυνατότητα διάχυσης του know how στο προσωπικό που θα αναλάβει την ανάπτυξη
  - Υπάρχουν διαθέσιμα τα τεχνολογικά εργαλεία που απαιτούνται για την ανάπτυξη
  - Υπάρχει το διαθέσιμο προσωπικό
  - Έχει το προσωπικό τις απαραίτητες δεξιότητες

CSD

Ανάλυση και πρόβλεψη κινδύνων (4/8)

- ***Feasibility study (2/2)***
  - Υπάρχει γνώση χρήσης των απαραίτητων τεχνολογικών εργαλείων
  - Υπάρχουν οι απαραίτητες διοικητικές δομές υποστήριξης της διαδικασίας παραγωγής
  - Υπάρχει δυνατότητας εξασφάλισης μίας ομάδας ανάπτυξης από την αρχή έως το τέλος της παραγωγής
  - Είναι οι όλοι χρηματοδοτικοί πόροι διαθέσιμοι ή εξασφαλισμένοι εκ των προτέρων

CSD

Ανάλυση και πρόβλεψη κινδύνων (5/8)

- ***Cost estimate (1/2)***
  - Εάν έχουν κατασκευαστεί τέτοια συστήματα στο πρόσφατο παρελθόν και υπάρχουν στοιχεία διαθέσιμα, είναι εφικτή μία καλή πρόβλεψη προϋπολογισμού
  - Ειδάλλως εκτιμήσεις μπορεί να γίνουν αρχικά για τις φάσεις που δεν εμπλέκουν σχεδίαση και υλοποίηση λογισμικού, ενώ για την υλοποίηση απαιτείται:
    - αρχιτεκτονική σχεδίαση και κατάτμηση
    - χρονοδιάγραμμα με παραδοτέα
    - διαμοιρασμός υλοποίησης
    - δυναμική αναπροσαρμογή βάσει παρακολούθησης

CSD

Ανάλυση και πρόβλεψη κινδύνων (6/8)

- ***Cost estimate (2/2)***
  - **Θεωρούμε περίπου 25% του χρόνου αφιερωμένο στην αντιμετώπιση λαθών** (εξαρτάται από το μέγεθος των συστημάτων)
  - Η μεγαλύτερη πρόκληση είναι η καλή πρόβλεψη του απαιτούμενου χρόνου ανάπτυξης
  - Σχεδόν όλες οι συστηματικές προβλέψεις αποδεικνύονται οπτιμιστικές
    - ***η ανάπτυξη τείνει να καθυστερεί πάντα περισσότερο από ότι αρχικά υπολογίζουμε***

CSD

Ανάλυση και πρόβλεψη κινδύνων (7/8)

- ***Progress monitoring (1/2)***
  - Αποκλίσεις από το χρονοδιάγραμμα
  - Καταγραφή στατιστικών στοιχείων
    - Ρυθμός ολοκλήρωσης των features
    - Χρόνος ανάπτυξης τμήματος ανά άτομο
    - Λάθη ανά εβδομάδα
    - Χρόνος διόρθωσης κάθε λάθους
    - Ρυθμός παραγωγής κώδικα
  - Απόδοση προσωπικού
    - Τμήματα, γραμμές κώδικα ανά εβδομάδα
    - Πρόκληση ή επιδιόρθωση λαθών

CSD

Ανάλυση και πρόβλεψη κινδύνων (8/8)

- ***Progress monitoring (2/2)***
  - Αλλαγές στην σχεδίαση και υλοποίηση
    - Σχεδίαση και υλοποίηση που δεν χρησιμοποιείται (ζημία)
    - Υποχώρηση του χρονοδιαγράμματος
    - Μεταβολές παραδοτέων και αναγκαίων πόρων
  - Εξωγενείς αλλαγές και δυναμική αναδιοργάνωση
    - Αλλαγές τεχνολογίας και απαρχαίωση
    - Αλλαγές εργαλείων και αυτοματοποιήσεις
    - Μεταβολές προσωπικού
    - Μεταβολές χρηματοδότησης
    - Ανταγωνισμός και νέα προϊόντα

CSD

Σχεδιαστικοί στόχοι στον κώδικα

- ***Modularity***
  - Καλής ποιότητας οργάνωση των συναρτήσεων σε τμήματα
- ***Reusability***
  - Δυνατότητα να γράφουμε γενικό κώδικα που μπορεί να επαναχρησιμοποιηθεί σε παρόμοιες περιπτώσεις
- ***Robustness***
  - Λειτουργική αξιοπιστία, αποφυγή σφαλμάτων εκτέλεσης, γρήγορος ή άμεσος εντοπισμός λαθών
- ***Quality***
  - Καλή ποιότητα κώδικα που διευκολύνει διορθώσεις, αλλαγές και επεκτάσεις