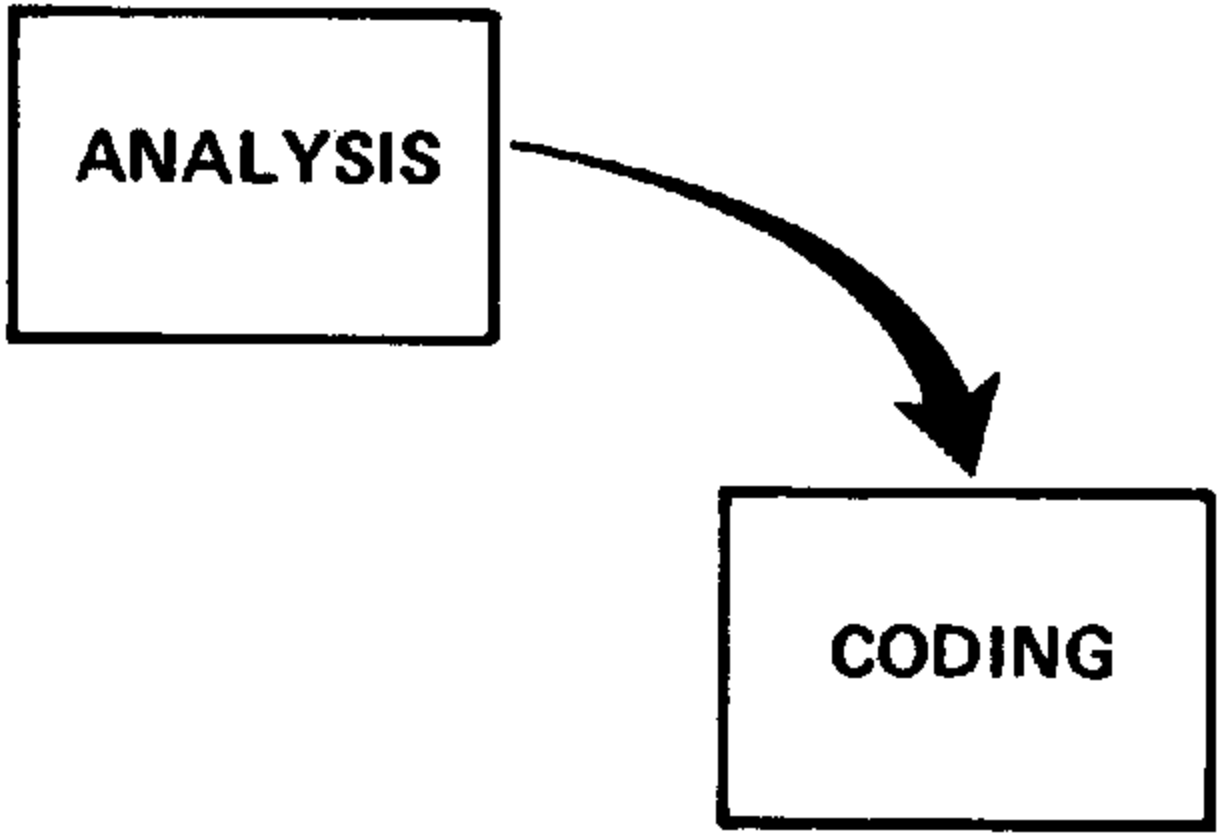
01 Εισαγωγή

Σχολή Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών
Εθνικό Μετσόβιο Πολυτεχνείο

Χειμερινό εξάμηνο 2017-18

Δρ. Κώστας Σαΐδης (saiko@di.uoa.gr)

Διαδικασία ανάπτυξης μικρού και απλού λογισμικού



W.W. Royce, www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf

Ανάπτυξη μεγάλου και σύνθετου λογισμικού;

Τεχνολογία Λογισμικού

Μεθοδολογίες, τεχνολογίες, εργαλεία και πρακτικές για τη βέλτιστη υλοποίηση σύνθετων έργων λογισμικού

Τεχνολογία λογισμικού

- Μεγάλα έργα
- Με πολλές και μεγάλες ομάδες
- Με πολύπλοκες απαιτήσεις
- Με οικογένειες εφαρμογών
- Με πολλές παράλληλες αλλαγές και εκδόσεις
- Που χρησιμοποιούνται για πολλά χρόνια

Τεχνολογία Λογισμικού

Γιατί μας απασχολεί η τεχνολογία λογισμικού;

Γιατί το λογισμικό, αν και κατέχει πλέον, σημαίνοντα ρόλο σε όλες σχεδόν τις δραστηριότητες της ανθρωπότητας είναι πολύ δύσκολο να αναπτυχθεί "σωστά"!

Σωστά;

- Εντός προϋπολισμού
- Έγκαιρα
- Ικανοποιώντας πλήρως τους χρήστες του
- Χωρίς λάθη και προβλήματα στη λειτουργία του
- Να είναι αξιόπιστο, να είναι εύκολο να συντηρηθεί, να επεκταθεί, κτλ.

Τι το ιδιαίτερο έχει η ανάπτυξη λογισμικού;

Γιατί είναι δύσκολο να γίνει "σωστά";

> Most software today is very much like an Egyptian pyramid with millions of bricks piled on top of each other, with no structural integrity, but just done by brute force and thousands of slaves.”
>
> Alan Kay

Alan Kay

- "Alan Curtis Kay is an American computer scientist ... best known for his pioneering work on object-oriented programming and windowing graphical user interface design." (Wikipedia).
- Βραβείο Turing (2003).

Ένας (πιθανός) λόγος

Στα μεγάλα έργα λογισμικού εμπλέκονται πολλοί συμμετέχοντες:

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

Επομένως

Κάθε ένας από τους συμμετέχοντες έχει διαφορετικές εμπειρίες, βιώματα, αφετηρίες, στόχους και επιδιώξεις από την υλοποίηση του έργου.

Είναι δύσκολο να επιτευχθεί ένας κοινός τόπος, σωστά ιεραρχημένος για το όφελος του έργου και, συνάμα, ικανοποιητικός για όλους τους συμμετέχοντες.

Δεύτερος (πιθανός) λόγος

> Most software estimates are performed at the beginning of the life cycle. This makes sense until we realize that estimates are obtained before the requirements are defined and thus before the problem is understood. Estimation, therefore, usually occurs at the wrong time.
>
> Robert Glass

Robert Glass

- "Robert L. (Bob) Glass (born 1932) is an American software engineer and writer, known for his works on software engineering, especially on the measuring of the quality of software design and his studies of the state of the art of software engineering research." (Wikipedia)

Robert L. Glass, "Facts and Fallacies of Software Engineering", Addison Wesley, 2002, 0-321-11742-5.

Τρίτος (πιθανός) λόγος

Σε επίπεδο υλοποίησης, το κύριο πρόβλημα είναι η πολυπλοκότητα (complexity):

- τυχαία πολυπλοκότητα (accidental)
- ουσιώδης πολυπλοκότητα (essential)

Ακόμα κι αν γίνουν σωστές εκτιμήσεις και βρεθεί κοινός τόπος, ένα έργο λογισμικού μπορεί να αποτύχει λόγω της ουδιώδους πολυπλοκότητάς του.

Πολυπλοκότητα

> The function of good software is to make the complex appear to be simple.
>
> Grady Booch

Grady Booch

"Grady Booch (born February 27, 1955) is an American software engineer, best known for developing the Unified Modeling Language (UML) with Ivar Jacobson and James Rumbaugh. He is recognized internationally for his innovative work in software architecture, software engineering, and collaborative development environments." (Wikipedia)

Τι το ιδιαίτερο έχει η ανάπτυξη λογισμικού;

Με τι μοιάζει και σε τι διαφέρει από τα άλλα ανθρώπινα έργα και προϊόντα;

> Programming is an unnatural act.
>
> Alan Perlis

Alan Perlis

- "Alan Jay Perlis (April 1, 1922 – February 7, 1990) was an American computer scientist and professor at Yale University known for his pioneering work in programming languages and the first recipient of the Turing Award [**To 1966!**]" (Wikipedia).

Alan J. Perlis, "Epigrams on Programming", ACM SIGPLAN Notices, Vol. 17, Issue 9, Sep. 1982, p. 7-13.
www.cs.yale.edu/~perlis-alan/quotes.html

Αν το σκεφτείτε

Το λογισμικό:

- δεν υπάρχει (δεν έχει φυσική υπόσταση)
- δεν τελειώνει/οριστικοποιείται ποτέ (εκτός από ένα/δυο 😄)
- πρέπει να λειτουργεί/διατηρείται σε πολλές παράλληλες εκδόσεις

Ο νόμος του Wirth

Η ταχύτητα με την οποία το λογισμικό γίνεται πιο αργό είναι μεγαλύτερη από την ταχύτητα με την οποία το υλικό γίνεται πιο γρήγορο.

> Were it not for a thousand times faster hardware, modern software would be utterly unusable.
>
> Niklaus Wirth

Niklaus Wirth

- "Niklaus Emil Wirth (born 15 February 1934) is a Swiss computer scientist, best known for designing several programming languages, including Pascal, and for pioneering several classic topics in software engineering." (Wikipedia)
- Βραβείο Turing (1984).

N. Wirth, "A Plea for Lean Software", IEEE Computer, Vol 28, Issue 2, Feb. 1995, p. 64-68.

Ωχ!

> The most amazing achievement of the computer software industry is its continuing cancellation of the steady and staggering gains made by the computer hardware industry.
>
> Henry Petroski

- "Henry Petroski is an American engineer specializing in failure analysis. A professor both of civil engineering and history at Duke University, he is also a prolific author." (Wikipedia).

Henry Petroski, "To Engineer Is Human: The Role of Failure in Successful Design", St. Martin's Press, 1985.

Σφαίρα από ασήμι

> So we hear desperate cries for a silver bullet, something to make software costs drop as rapidly as computer hardware costs do. But, as we look to the horizon of a decade hence, we see no silver bullet. There is no single development, in either technology or management technique, which by itself promises even one order of magnitude improvement in productivity, in reliability, in simplicity.
>
> Fred Brooks

Fred Brooks

- "Frederick Phillips Brooks, Jr. (born April 19, 1931) is an American computer architect, software engineer, and computer scientist, best known for managing the development of IBM's System/360 family of computers and the OS/360 software support package, then later writing candidly about the process in his seminal book The Mythical Man-Month." (Wikipedia).
- Βραβείο Turing (1999).

Fred P. Brooks, "No Silver Bullet — Essence and Accident in Software Engineering", Proceedings of the IFIP Tenth World Computing Conference, April 1987, p. 1069-1076. http://www.cs.nott.ac.uk/~pszcah/G51ISS/Documents/NoSilverBullet.html

Fred P. Brooks, "The Mythical Man Month", Addison-Wesley, 1975, 0-201-00650-2.

Ο Μυθικός ανθρωπο-μήνας

Η προσθήκη ανθρωπο-προσπάθειας σε ένα αργοπορημένο έργο λογισμικού το καθυστερεί ακόμα περισσότερο!

Περιεχόμενα διαλέξεων

Κύκλος Ζωής του Λογισμικού

- Καταγραφή και ανάλυση απαιτήσεων.
- Σχεδιασμός.
- Υλοποίηση.
- Επαλήθευση και επικύρωση.
- Εγκατάσταση, έλεγχος, παραμετροποίηση και ολοκλήρωση λογισμικού στο παραγωγικό του περιβάλλον.
- Συντήρηση και επέκταση.

Μοντέλα ανάπτυξης λογισμικού

- Ακολουθιακή διαδικασία (sequential)
- Μοντέλο Καταρράκτη (waterfall)
- Επαναληπτική διαδικασία (iterative)
- Αυξητική διαδικασία (incremental)
- Ευέλικτη διαδικασία (agile)

Μεθοδολογίες ανάπτυξης λογισμικού

- Ταχεία Πρωτοτυποίηση (Rapid prototyping).
- «Ακραίος Προγραμματισμός» (Extreme Programming).
- Ανάπτυξη βασισμένη σε ελέγχους (Test-driven Development).
- Συνεχής παράδοση (Continuous Delivery)
- Ανάπτυξη / Λειτουργία («DevOps»).

Ανάλυση απαιτήσεων

- Ανάλυση και μοντελοποίηση απαιτήσεων
- Σύνταξη προδιαγραφών και παραδοτέων

Σχεδιασμός Λογισμικού

- Βασικές αρχές
- Αντικειμονοστρεφής σχεδιασμός συστημάτων
- Συστατικά λογισμικού

Γλώσσα μοντελοποίησης UML

- Διαγράμματα κλάσεων (Class diagrams)
- Διαγράμματα ακολουθιών (Sequence diagrams)
- Διαγράμματα δραστηριοτήτων (Activity diagrams)
- Διαγράμματα σεναρίων χρήσης (Use-case diagrams)
- κτλ.

Πρότυπα Σχεδίασης (Design Patterns)

- Κατασκευαστικά πρότυπα (Creational patterns)
- Δομικά πρότυπα (Structural patterns)
- Πρότυπα συμπεριφοράς (Behavioral patterns)

Αρχιτεκτονική Λογισμικού

- Αρχιτεκτονικός σχεδιασμός σύνθετων κατανεμημένων συστημάτων
  - Βασικές έννοιες
  - Έμφαση σε εφαρμογές διαδικτύου
  - Αρχιτεκτονική REST (Representation State Transfer)
- Ανάπτυξη RESTful Application Programming Interfaces (APIs)
- Θέματα ασφάλειας
- Θέματα απόδοσης

Σχεδιασμός διεπαφής χρήστη

- Ευχρηστία, διαδραστικότητα και αποκρισιμότητα
- Σύγχρονες μεθοδολογίες ανάπτυξης διεπαφής χρήστη
  - Πρότυπα σχεδίασης Model-View-Controller και Observable
  - Έμφαση σε τεχνολογίες εφαρμογών διαδικτύου (HTML5, CSS3, Javascript)
  - Εφαρμογές μιας σελίδας (Single-page applications)
  - Ασύγχρονες τεχνικές (AJAX, Promises, Reactive frameworks)

Διοίκηση και Διαχείριση Έργου Λογισμικού

- Διοίκηση έργου (γενικά)
  - Εισαγωγή και βασικές έννοιες
  - Εκτίμηση κόστους έργου
- Διοίκηση ομάδας ανάπτυξης λογισμικού
  - Συνήθεις ρόλοι σε μια ομάδα ανάπτυξης λογισμικού
  - Αλληλεπιδράσεις μεταξύ ρόλων
- Καλές διεθνείς πρακτικές

Τεχνικές και Εργαλεία Διοίκησης, Παρακολούθησης και Ελέγχου της Ανάπτυξης Λογισμικού

- Έλεγχος εκδόσεων (Version Control)
  - Έμφαση στο σύστημα Git (το πλέον διαδεδομένο Version Control System)
- Αυτοματισμός διαδικασίας «χτισίματος» λογισμικού (Build automation)
  - Έμφαση στο σύστημα Gradle (χρησιμοποιείται στην ανάπτυξη Android εφαρμογών)
- Ανάλυση προγράμματος (program analysis) και αυτόματος εντοπισμός σφαλμάτων (bug detection)

- Σενάρια ελέγχου
  - Unit testing, Regression testing, Functional testing, Integration testing, test coverage
- Συνεχής ολοκλήρωση (Continuous integration)
- Αξιοπιστία λογισμικού
- Η έννοια του «τεχνικού/σχεδιαστικού χρέους» (technical debt).

Διαχείριση συστατικών του λογισμικού

- Συστατικά λογισμικού και αλληλεξαρτήσεις τους (software components and dependencies)
- Αποθήκες συστατικών λογισμικού (Software artifact repositories)
- Διαχείριση εκδόσεων λογισμικού (software releases)
- Μέθοδοι αρίθμησης εκδόσεων (versioning schemes)

Αν προλάβουμε

- Γλώσσες ειδικού σκοπού (Domain specific languages)
- Reflection
- Aspect-oriented programming
- Metaprogramming

Πριν ξεκινήσουμε

Μια τελευταία οπτική

Ανάπτυξη λογισμικού: τέχνη ή επιστήμη;

> When CACM began publication in 1959, the members of ACM's Editorial Board made the following remark as they described the purposes of ACM's periodicals: "If computer programming is to become an important part of computer research and development, a transition of programming from an art to a disciplined science must be effected." Such a goal has been a continually recurring theme during the ensuing years; for example, we read in 1970 of the "first steps toward transforming the art of programming into a science". Meanwhile we have actually succeeded in making our discipline a science, and in a remarkably simple way: merely by deciding to call it "computer science."
>
> Donald Knuth

Donald Knuth

- Donald Ervin Knuth is an American computer scientist, mathematician, and professor emeritus at Stanford University. He is the author of the multi-volume work The Art of Computer Programming. He contributed to the development of the rigorous analysis of the computational complexity of algorithms and systematized formal mathematical techniques for it ... Knuth is the creator of the TeX computer typesetting system, the related METAFONT font definition language and rendering system, and the Computer Modern family of typefaces.
- Βραβείο Turing (1974).

Donald E. Knuth, "Computer Programming as an Art", Communications of the ACM, Vol. 7, Issue 12, Dec. 1974, p. 667-673.

(Η ομιλία του D. Knuth κατά τη λήψη του βραβείου Turing το 1974)

www.cs.bilkent.edu.tr/~canf/knuth1974.pdf

Κατά τη γνώμη του Knuth

> To summarize: We have seen that computer programming is an art, because it applies accumulated knowledge to the world, because it requires skill and ingenuity, and especially because it produces objects of beauty. A programmer who subconsciously views himself as an artist will enjoy what he does and will do it better. Therefore we can be glad that people who lecture at computer conferences speak about the state of the Art.
>
> Donald Knuth

Άλλη μια γνώμη σχετικά

> To put it in another way: it is my purpose to transmit the importance of good taste and style in programming, the specific elements of style presented serve only to illustrate what benefits can be derived from "style" in general. In this respect I feel akin to the teacher of composition at a conservatory: he does not teach his pupils how to compose a particular symphony, he must help his pupils to find their own style and must explain to them what is implied by this. (It has been this analogy that made me talk about "The Art of Programming".)
>
> E.W. Dijkstra

Edsger Dijkstra

- "One of the most influential members of computing science's founding generation, Dijkstra helped shape the new discipline from both an engineering and a theoretical perspective. Many of his papers are the source of new research areas. Several concepts and problems that are now standard in computer science were first identified by Dijkstra or bear names coined by him." (Wikipedia)
- Βραβείο Turing (1972).

E.W. Dijkstra, "EWD316: A Short Introduction to the Art of Programming", August 1971.

https://www.cs.utexas.edu/users/EWD/transcriptions/EWD03xx/EWD316.html

Και μια τελευταία

> What we can say with some confidence is that these are the glory days of hacking. In most fields the great work is done early on. The paintings made between 1430 and 1500 are still unsurpassed. Shakespeare appeared just as professional theater was being born, and pushed the medium so far that every playwright since has had to live in his shadow.
>
> ...
>
> Over and over we see the same pattern. A new medium appears, and people are so excited about it that they explore most of its possibilities in the first couple generations. Hacking seems to be in this phase now. Painting was not, in Leonardo's time, as cool as his work helped make it. How cool hacking turns out to be will depend on what we can do with this new medium.
>
> Paul Graham

Paul Graham

- "Paul Graham is an English computer scientist, venture capitalist, and essayist. He is known for his work on Lisp, for co-founding Viaweb (later renamed "Yahoo! Store"), and for co-founding the Y Combinator seed capital firm." (Wikipedia).

Paul Graham, "Hackers and Painters", O'Reilly Media, 2004, 0-596-00662-4.

Εδώ το διάσημο ομώνυμο essay:
http://www.paulgraham.com/hp.html

Να θυμάστε ότι

> The best programmers are up to 28 times better than the worst programmers, according to "individual differences" research.
>
> Robert Glass