Προηγμένο Λογισμικό Υπηρεσιών Ιστού

Ευάγγελος Σακκόπουλος

Επίκουρος Καθηγητής

Μεταπτυχιακό Πρόγραμμα Εαρινού Εξαμήνου
Παρασκευή 6-9μμ

Εξέταση με απαλλακτική εργασία

Παράδοση στο τέλος της εξεταστικής
Παρακολουθείτε το σύστημα gunet2.cs.unipi.gr

https://gunet2.cs.unipi.gr/courses/TMH111/index.php

2019-2020

Αντικείμενο

- Προηγμένη Τεχνολογία Λογισμικού
- Αρχές Αντικειμενοστραφούς Προγραμματισμού
- Αντικειμενοστραφή Πρότυπα Σχεδίασης Λογισμικού (Software Design Patterns)
- Εξομοίωση Εξάσκησης σε Πρότυπα Σχεδίασης Λογισμικού
- Προηγμένο Λογισμικό Υπηρεσιών Ιστού & Πρότυπα Σχεδίασης (Cloud Design Patterns)
- Εξομοίωση Εξάσκησης σε Πρότυπα Σχεδίασης Προηγμένου Λογισμικού Υπηρεσιών Ιστού

Τεχνολογία Λογισμικού

- **Σχεδίαση,**
- **Υλοποίηση και**
- **Συντήρηση**

μεγάλων συστημάτων λογισμικού

Παλαιός Ορισμός

Κλάδος της Πληροφορικής που ασχολείται με τη μελέτη και ανάπτυξη τεχνικών για την παραγωγή λογισμικού που ικανοποιεί τις προδιαγραφές του, με την καλύτερη δυνατή ποιότητα

Τροποποίηση ορισμού ….

Κλάδος της Πληροφορικής που ασχολείται με τη μελέτη και ανάπτυξη τεχνικών για την παραγωγή λογισμικού που ικανοποιεί τις προδιαγραφές του, με την καλύτερη δυνατή ποιότητα,

- παραδίδεται μέσα σε προδιαγεγραμμένα χρονικά όρια και
- το κόστος ανάπτυξής του βρίσκεται μέσα σε προδιαγεγραμμένα όρια

Ειπαν….

Software Engineering is:

David Parnas: "the multi-person construction of multi-version software"

Ghezzi: "the application of engineering to software"

IEEE: "the application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software"

Roberts: "the discipline of writing programs that can be understood and maintained by others"

in contrast to:

"Programming is the discipline of writing programs that cannot be understood and maintained by anyone else than the author"

Software Crisis

Κατά κανόνα, η ανάπτυξη μεγάλων έργων λογισμικού παρουσιάζει προβλήματα:

- **υπερβάσεις στο χρονοδιάγραμμα**
- **υπερβάσεις στον προϋπολογισμό**
- **παραγόμενο προϊόν κακής ποιότητας**
- **πολυδάπανο στη συντήρησή του**

1979:

από έργα 6.8 εκατομμυρίων δολλαρίων
47% πληρώθηκε αλλά δεν παραδόθηκε προς χρήση
29% παραδόθηκε αλλά δεν χρησιμοποιήθηκε
19% τροποποιήθηκε μετά την παράδοση
3% χρησιμοποιήθηκε με μικρές αλλαγές
2% χρησιμοποιήθηκε όπως παραδόθηκε

Software Crisis

Αρχές '80: το IRS ανέθεσε στην Sperry Corp την ανάπτυξη συστήματος για την αυτόματη επεξεργασία φορμών

- **το 1985 απαιτήθηκαν επιπλέον $90 million για τον εμπλουτισμό του αρχικού έργου (αξίας $103 million)**
- **το σύστημα δεν βελτιώθηκε αισθητά**
- **Στο Κογκρέσο κατηγορήθηκε ότι στο έργο είχαν χαθεί συνολικά $4 billion**

Αρχές '90: Therac-25, σύστημα ελέγχου ακτινοβολίας ασθενών

- **Κακή σχεδίαση του λογισμικού: Οδήγησε στο θάνατο αρκετών ασθενών από υπερβολικές δόσεις ακτινοβολίας**

Software Crisis

Software Hall of Shame

Year	Company	Outcome - Costs in US $
2005	FBI	Virtual Case File abandoned after 170 million spent
2004	UK Revenue	Software errors -> 3.45 billion tax overpayment
2004	HP	Problems with ERP system -> 160 million loss
2002	McDonald's Corp	Information processing system canceled after 170million spent
2001	Nike Inc.	Problems with supply chain management -> 100 million loss
2000	Kmart Corp.	1.4 billion IT modernization effort -> declared bankruptcy
1997	IRS (USA)	Tax modernization effort canceled after 4 billion is spent
1993	London Stock Exchange	Taurus stock settlement system abandoned after 600 million spent

Συντηρητική Εκτίμηση: Στις ΗΠΑ, απώλειες 60 – 70 δις δολ / έτος

Software Crisis

Γιατί αποτυγχάνουν τα έργα λογισμικού τόσο συχνά?

- **ανεπαρκής προσδιορισμός απαιτήσεων** -> προβληματική σχεδίαση
- **μη ρεαλιστικοί στόχοι του project**
- **μη ακριβείς εκτιμήσεις απαιτούμενων πόρων**
- **κακή αναφορά προόδου**
- **ελλιπής χειρισμός ρίσκου**
- **κακή επικοινωνία μεταξύ πελατών, προγρ/στών, χρηστών**
- **έλλειψη εμπειρίας με τεχνολογία**
- **αδυναμία χειρισμού πολυπλοκότητας**

Ιδιαιτεροτητες εργων λογιςμικου

Διαφέρουν τα έργα λογισμικού από τα άλλα έργα ?

Test 1: Έστω ότι ένα έργο λογισμικού έχει καθυστερήσει σε σχέση με το χρονοδιάγραμμα του. Αν είσαστε ο project manager με τι τρόπους μπορείτε να το επιταχύνετε ?

Αν απαντήσατε με αύξηση των ατόμων ή με χρήση νέων εργαλείων ….. wrong answer ! (Mythical Man-Month)

Test 2: Έστω ότι είστε ο project manager σε ένα έργο λογισμικού (μη έχοντας σχέση με προγραμματισμό). Πώς θα παρακολουθήσετε την πορεία του ?

Προφανώς εξαρτάστε από τις αναφορές των υπολοίπων !!

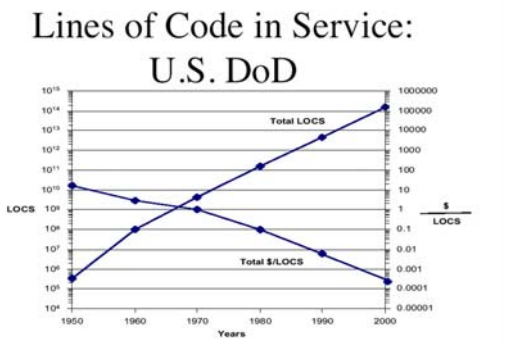
Test 3: Πουλάτε ένα προϊόν λογισμικού και σας ρωτούν να προσδιορίσετε την ποιότητα του. Τι θα αναφέρατε ?

Προφανώς δεν υπάρχουν σαφή χαρακτηριστικά ποιότητας !!!

Software Engineering

- Η οικονομία ΟΛΩΝ των ανεπτυγμένων κρατών βασίζεται σε software
- Ολοένα και περισσότερα συστήματα ελέγχονται από software
- Η τεχνολογία λογισμικού ασχολείται με θεωρία, μεθόδους και εργαλεία για επαγγελματική ανάπτυξη λογισμικού
- Οι δαπάνες για ανάπτυξη λογισμικού αποτελούν σημαντικό ποσοστό του ΑΕΠ σε όλες τις ανεπτυγμένες χώρες

Software Engineering



Intenet

Κόστος Λογισμικού

- Το κόστος του λογισμικού συχνά κυριαρχεί στο κόστος ενός συστήματος και είναι υψηλότερο από το κόστος του αντίστοιχου υλικού.
- Κόστος υλικού ↓ Κόστος λογισμικού ↑ (Καμπύλη S)
- Το λογισμικό κοστίζει περισσότερο να συντηρηθεί παρά να υλοποιηθεί. Για συστήματα με μεγάλο χρόνο ζωής το κόστος συντήρησης είναι πολλαπλάσιο του κόστους υλοποίησης.

Κόστος Λογισμικού

Δεδομένα:

USA : $ 500 billion -> software production

Παγκοσμίως: 1994: $140 billion -> 2000: $ 800 billion (15% incr. each year)

IBM: Κόστος για την ανάπτυξη του OS/2 : $200 million

Αποστολές στο διάστημα μεταξύ 1960 – 70 (software costs): $1 billion

Κόστος Λογισμικού

Σε τι συνίσταται το κόστος λογισμικού ? (Τι πληρώνουμε)

Μισθούς προγραμματιστών ως επί το πλείστον !!

Ποια η παραγωγικότητα του μέσου προγραμματιστή ?

10 – 20 προτάσεις γλώσσας προγραμματισμού /ημέρα

Ωστόσο:

Διαφορές μεταξύ προγραμματιστών μέχρι 1 προς 20

Application software γράφεται γρηγορότερα από system software

Κόστος Λογισμικού

Η εκτίμηση του κόστους που απαιτείται για τη συγγραφή λογισμικού πολύ δύσκολη:

50% under-estimation by managers (κόστος, χρονοδιάγραμμα, staffing, resources)

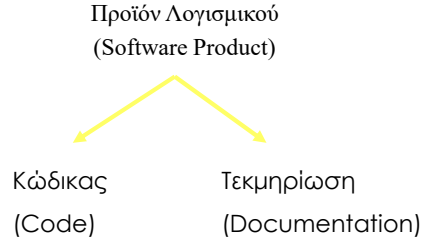
Κόστος Λογισμικού

Πώς σχηματίζεται η κοινή γνώμη για το κόστος λογισμικού:

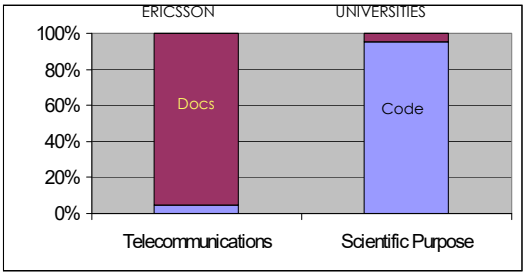
- Παράγοντας ‘’rock’n’roll’’: Αν αγοράζει ένα stereo για $200 δεν περιμένεις το CD να κοστίζει $2,000.

  Το ίδιο υποθέτουν για τους υπολογιστές
- Σύνδρομο ‘’teenager in the bedroom’’: Πολλοί φοιτητές/μαθητές γράφουν προγράμματα σπίτι. Γονείς: Αφού είναι τόσο εύκολο, γιατί να κοστίζει πολύ ?
- Πακέτα λογισμικού με χαμηλό κόστος δίνουν την εντύπωση ότι το κόστος ανάπτυξης είναι εξίσου χαμηλό
- Ορισμένες εφαρμογές (VBasic, Access, Excel) δίνουν την εντύπωση ότι είναι εύκολο να γράψεις λογισμικό

Προϊόντα Λογισμικού



Προϊόντα Λογισμικού



Προϊόντα Λογισμικού

- Προγράμματα Γενικής Χρήσης
  - Αυτόνομα συστήματα που υλοποιούνται από μεγάλους συνήθως κατασκευαστές και πωλούνται στην αγορά σε οποιονδήποτε αγοραστή. (Microsoft Windows XP !!!)
- Εξειδικευμένα προγράμματα
  - Συστήματα που υλοποιούνται αποκλειστικά για έναν πελάτη από μία εταιρεία. (Λογισμικό ΟΤΕ)
- Το μεγαλύτερο ποσοστό του τζίρου αφορά γενικής χρήσης προγράμματα αλλά η μεγαλύτερη προσπάθεια καταβάλλεται στα εξειδικευμένα προγράμματα (περισσότερες ανθρωποώρες)

Προϊόντα Λογισμικού

Άλλος διαχωρισμός:

Λογισμικό Συστημάτων: Λογισμικό που λειτουργεί ως εργαλείο για τη δημιουργία λογισμικού εφαρμογών (Λειτουργικά, Βάσεις Δεδομένων, Compilers)

Λογισμικό Εφαρμογών: Λογισμικό που εκτελεί κάποια χρήσιμη εργασία (παιχνίδια, επεξεργαστές κειμένου, λογισμικό αεροσκαφών)

Χαρακτηριστικά Προϊόντων Λογισμικού

- Ορθότητα (Correctness)
  - Ένα πρόγραμμα είναι *λειτουργικώς ορθό* όταν συμπεριφέρεται σύμφωνα με τις καταγεγραμμένες λειτουργικές απαιτήσεις.
- Αξιοπιστία (Reliability)
  - Το λογισμικό θα πρέπει να μην προκαλεί φυσική ή οικονομική καταστροφή στην περίπτωση λάθους. (Η πιθανότητα το λογισμικό να συμπεριφέρεται σωστά σε ένα συγκεκριμένο χρονικό διάστημα)
- Αποδοτικότητα (Performance)
  - Το πρόγραμμα δεν θα πρέπει να κάνει αλόγιστη χρήση των πόρων του συστήματος
- Ευχρηστία (Usability)
  - Το πρόγραμμα θα πρέπει να έχει ένα εύχρηστο περιβάλλον επικοινωνίας με το χρήστη και κατάλληλη τεκμηρίωση

Ghezzi

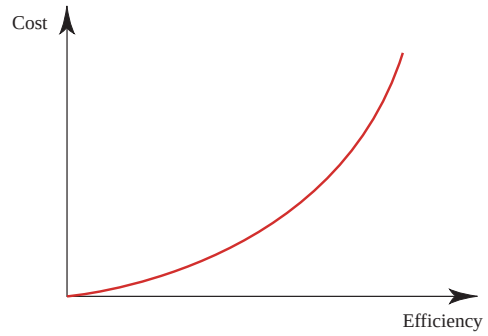
Χαρακτηριστικά Προϊόντων Λογισμικού

- Ευελιξία – Συντηρησιμότητα (Maintainability)
  - Σε περίπτωση αλλαγής των απαιτήσεων ένα πρόγραμμα θα πρέπει να μπορεί να εξελίσσεται για την κάλυψη των.
- Ελεγξιμότητα (Verifiability)
  - Οι ιδιότητες ενός συστήματος λογισμικού θα πρέπει να μπορούν να ελεγχθούν εύκολα. Π.χ. η λειτουργική ορθότητα, ή η απόδοση πρέπει να μπορούν να ελεγχθούν με χρήση προσομοίωσης, ή μέσω τυπικών μεθόδων
- Δυνατότητα Επαναχρησιμοποίησης (Reusability)
  - Ένα πρόγραμμα είναι επαναχρησιμοποιήσιμο εάν μπορεί να χρησιμοποιηθεί (ενδεχομένως με μικρές αλλαγές) για την ανάπτυξη άλλου προϊόντος λογισμικού.
- Φορητότητα (Portability)
  - Το λογισμικό είναι φορητό εάν μπορεί να εκτελεστεί σε διαφορετικά περιβάλλοντα (λειτουργικά συστήματα, εξάρτηση από βάσεις).

Χαρακτηριστικά Προϊόντων Λογισμικού

- Η σχετική σημασία κάθε χαρακτηριστικού εξαρτάται από το πρόγραμμα και το περιβάλλον στο οποίο πρόκειται να χρησιμοποιηθεί
- Σε ορισμένες περιπτώσεις κυριαρχούν ορισμένα χαρακτηριστικά
  - Σε συστήματα ασφαλείας για παράδειγμα κύριο χαρακτηριστικό αποτελεί η αξιοπιστία
  - Σε real-time συστήματα απαιτείται υψηλή απόδοση
  - Σε portable συστήματα η χαμηλή κατανάλωση ενέργειας
- Το κόστος αυξάνει εκθετικά αν απαιτείται κάποιο από τα χαρακτηριστικά σε υψηλά επίπεδα

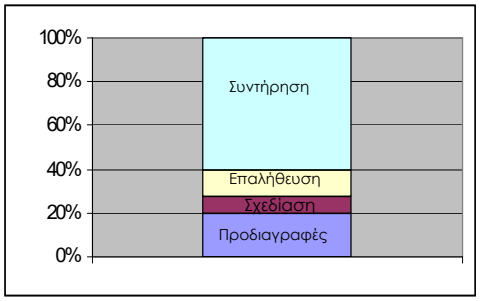
Κόστος Απόδοσης



Διαδικασία ανάπτυξης λογισμικού

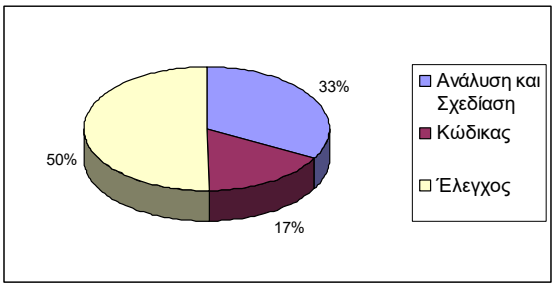
- Δομημένο σύνολο δραστηριοτήτων για την ανάπτυξη ενός συστήματος λογισμικού
  - Προδιαγραφές
  - Σχεδίαση
  - Επαλήθευση
  - Συντήρηση (Εξελικτική – Διορθωτική)
- Οι δραστηριότητες διαφέρουν ανάλογα με το σύστημα υπό ανάπτυξη και την εταιρεία
- Για τη διαχείριση της απαιτείται ακριβής μοντελοποίηση

Κατανομή Κόστους



Κατανομή Κόστους (Εξαιρώντας Συντήρηση)

Εμφανής η μεγάλη σημασία των λαθών στο λογισμικό



από το διάγραμμα εξαιρείται επίσης feasibility study, plan, καταγραφή)

Κόστος Λαθών

Ωστόσο, το πρόβλημα δεν είναι που είναι τα λάθη, αλλά πόσο κοστίζει η διόρθωσή τους

Το κόστος του λάθους αυξάνει όσο περισσότερο δεν εντοπίζεται

Συντήρηση

Γενικά το λογισμικό θεωρείται soft: εύκολο να αλλάξει

Περισσότερο μοιάζει με πάγο: Αν προσπαθήσεις να το αλλάξεις, πιο συχνά σπάει παρά λυγίζει !!!

"Σε ένα μεγάλο κτίριο πρέπει να αφαιρεθούν εκατοντάδες στρατηγικά τοποθετημένων τούβλων για να καταρρεύσει ένας τοίχος"….. σε ένα πρόγραμμα 100,000 γραμμών αρκούν 1-2 προβληματικές γραμμές για να προκληθούν μείζονα προβλήματα.

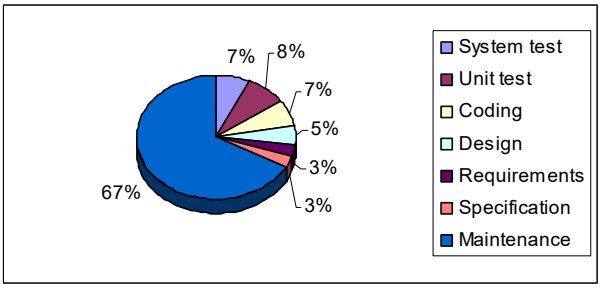
Το 1991, τμήμα του τηλεφωνικού δικτύου της AT&T κατέρρευσε, αφήνοντας 12 εκατομμύρια συνδρομητές χωρίς σύνδεση, λόγω ενός και μόνο λανθασμένου χαρακτήρα σε μία εντολή

Συντήρηση

Η συντήρηση θεωρείται συχνά μπελάς από όλους:

- **managers, διότι αυξάνει το κόστος, θέλει ανθρ. δυναμικό**
- **προγραμματιστές, διότι παρουσιάζει μικρότερο ενδιαφέρον από τη σχεδίαση**

Κατανομή Χρόνου σε ένα τυπικό έργο



Ο μπελάς της Συντήρησης

- Παγκοσμίως χρησιμοποιούνται 50 δισεκατομμύρια γραμμές Cobol: Λογισμικό παλαιότητας μέχρι και 30 ετών, χαμηλής ποιότητας τεκμηρίωση, εργαλεία ανύπαρκτα. Αντικατάσταση ? (Πανάκριβη). Μόνη λύση -> Συντήρηση
- UK: $ 1.5 million ξοδεύονται ετησίως σε συντήρηση
- Η αναγκαιότητα της συντήρησης έγινε εμφανής με το Millenium bug.

8 Θεμελιωδεις Εννοιες (Wasserman)

- Αφαίρεση
  - Απόκρυψη λεπτομερειών – Γενίκευση σε επίπεδα
- Σημειολογία
  - Τρόπος επικοινωνίας με την υπόλοιπη ομάδα
- Ανάπτυξη πρωτοτύπων
  - Ανάπτυξη μικρού ισοδύναμου με περιορισμένη λειτουργικότητα (executable σε embedded συστήματα)
- Αρχιτεκτονική Λογισμικού
  - Περιγραφή του συστήματος ως σύνολο μονάδων και του τρόπου διασύνδεσής τους

Pfleeger

8 Θεμελιωδεις Εννοιες (Wasserman) ….. 2

- Διαδικασία Ανάπτυξης Λογισμικού
  - Προσήλωση και πιστή εφαρμογή δραστηριοτήτων και κανόνων
- Επαναχρησιμοποίηση
  - Χρησιμοποίηση τμημάτων (κώδικα, τεκμηρίωσης, ελέγχων) από προηγούμενα έργα
- Μέτρηση
  - ‘’You cannot control what you cannot measure’’ (DeMarco)
- Εργαλεία και Περιβάλλοντα
  - CASE tools

Διαδικασία ανάπτυξης λογισμικού - Χαρακτηριστικά

- Κατανόηση
  - Είναι η διαδικασία σαφώς καθορισμένη και κατανοητή ?
- Παρακολούθηση
  - Είναι δυνατόν να παρακολουθήσουμε εξωτερικά την πρόοδο της διαδικασίας ?
- Υποστήριξη
  - Είναι δυνατόν να υποστηριχθεί η διαδικασία με εργαλεία (π.χ. CASE tools) ?
- Αποδοχή
  - Είναι η διαδικασία αποδεκτή από αυτούς που συμμετέχουν σε αυτή ?

Διαδικασία ανάπτυξης λογισμικού - Χαρακτηριστικά

- Αξιοπιστία
  - Ανακαλύπτονται τα λάθη της διαδικασίας πριν μετατραπούν σε λάθη στο προϊόν ?
- Ανθεκτικότητα
  - Είναι δυνατόν να αντεπεξέλθει η διαδικασία σε μη αναμενόμενα προβλήματα ?
- Συντήρηση
  - Είναι δυνατόν να εξελιχθεί η διαδικασία για την κάλυψη μεταβαλλόμενων προδιαγραφών και απαιτήσεων ?
- Ταχύτητα
  - Πόσο γρήγορα παράγεται το προϊόν ?

Ανάπτυξη Λογισμικού

Τι απαιτείται για να προχωρήσουμε στην ανάπτυξη ενός έργου (λογισμικού ή όχι) ?

- **Σαφής διατύπωση των απαιτήσεων**
- **Ένα σύνολο τεχνικών και εργαλείων**
- **Ένα πλάνο**

  Το πλάνο καθορίζει:

  - Τι εργασίες θα εκτελεστούν

  - Με ποια σειρά

  - Ποια τα κριτήρια μετάβασης

  - Ποια τα ενδιάμεσα παραδοτέα

  - Πώς αξιολογείται η πορεία του έργου

Bell

Ανάπτυξη Λογισμικού - Μοντέλα

Ένα πλάνο είναι γνωστό ως μοντέλο διαδικασίας (process model)

Ετυμολογία:

Διαδικασία: Σύνολο διακριτών βημάτων

Μοντέλο: Αναπαράσταση της πραγματικότητας

Διαδικασία ανάπτυξης **υλικού** - Μοντέλο

- Προδιαγραφές – καθορισμός των απαιτήσεων και των περιορισμών του συστήματος
- Σχεδίαση – Παραγωγή ενός μοντέλου του συστήματος "επί χάρτου"
- Υλοποίηση – δημιουργία του συστήματος
- Επικύρωση / Επαλήθευση – Έλεγχος αν το σύστημα πληροί τις απαιτούμενες προδιαγραφές
- Εγκατάσταση – Διανομή του συστήματος στον πελάτη και εξασφάλιση της ορθής λειτουργίας του
- Συντήρηση – Διόρθωση σφαλμάτων στο σύστημα όταν ανακαλύπτονται

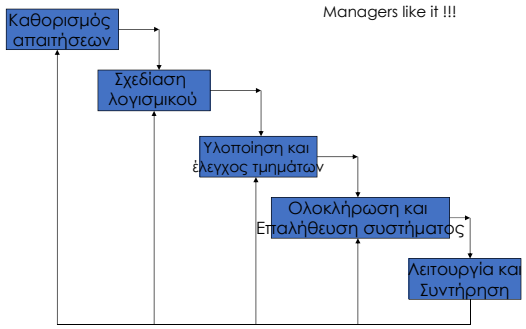
Διαδικασία ανάπτυξης **λογισμικού** - Μοντέλο

- Συνήθως, οι προδιαγραφές είναι ελλιπείς / προβληματικές. Παράδειγμα ξεκάθαρων απαιτήσεων ?
- …… ένας μεταγλωττιστής (compiler)
- Θολή διάκριση μεταξύ καθορισμού προδιαγραφών, σχεδίασης και υλοποίησης
- Το λογισμικό δεν φθείρεται – συντήρηση δεν σημαίνει αντικατάσταση εξαρτημάτων
- Διόρθωση στο ήδη εγκατεστημένο λογισμικό δεν είναι πάντοτε εφικτή

Γενικευμένα μοντέλα ανάπτυξης λογισμικού

- Κανένα (code-and-fix)
- Το μοντέλο καταρράκτη
  - Ξεχωριστές και διακριτές φάσεις καθορισμού προδιαγραφών και ανάπτυξης
- Εξελικτική ανάπτυξη
  - Ο καθορισμός των προδιαγραφών και η ανάπτυξη διαπλέκονται
- Τυπικός μετασχηματισμός
  - Ένα μαθηματικό μοντέλο του συστήματος μετασχηματίζεται βάσει κανόνων σε υλοποίηση
- Υλοποίηση με επαναχρησιμοποίηση
  - Το σύστημα συναρμολογείται από υπάρχοντα τμήματα

Μοντελο Καταρρακτη (1970)



Waterfall Model

- **Το μοντέλο καταρράκτη δεν κάνει υποθέσεις για τη μεθοδολογία και τους συμβολισμούς σε κάθε στάδιο**

Βασικές αρχές μοντέλου:

- **Ακολουθία βημάτων**
- **Κάθε βήμα είναι σαφώς καθορισμένο**
- **Κάθε βήμα καταλήγει στην δημιουργία προϊόντος (έγγραφο ή κώδικας)**
- **Κάθε προϊόν αποτελεί τη βάση για το επόμενο βήμα**
- **η ορθότητα κάθε προϊόντος μπορεί να ελεγχθεί**

Waterfall Model

Πλεονεκτήματα

- **Καλός διαχωρισμός του έργου σε απλούστερες φάσεις**
- **Κάθε φάση παράγει ένα σαφώς καθορισμένο παραδοτέο**

Μειονεκτήματα

- **Στην πράξη οι φάσεις αλληλεπικαλύπτονται**
- **Στην πράξη το μοντέλο δεν είναι γραμμικό: συχνά επιστρέφουμε στην προηγούμενη φάση**
- **Συχνά, αλλαγές σε κάποιο στάδιο επιβάλλουν την οπισθοχώρηση και πραγματοποίηση αλλαγών σε πολλά από τα προηγούμενα στάδια**
- **Ο πελάτης βλέπει τι τελικά αγοράζει πολύ αργά !!!**

Waterfall Model στην πραξη

Στρατιωτικό πρότυπο: MIL-STD-2167 A, MIL-STD-498

Περιλαμβάνει τις απαιτήσεις για την ανάπτυξη και αποδοχή misson-critical υπολογιστικών συστημάτων

Ειδικά για την ανάπτυξη λογισμικού:

- **b.1 software requirements analysis**
- **b.2 preliminary design**
- **b.3 detailed design**
- **b.4 coding and module testing**
- **b.5 computer software component integration and testing**
- **b.6 computer software configuration item testing**

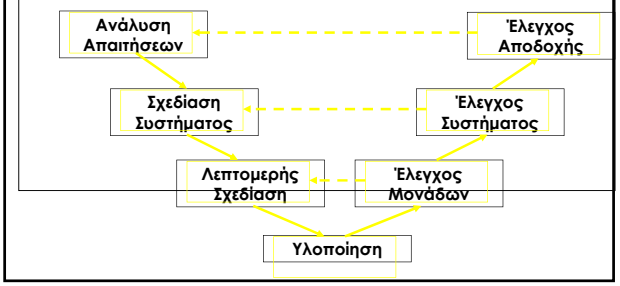
logical entities: modules

delivered software

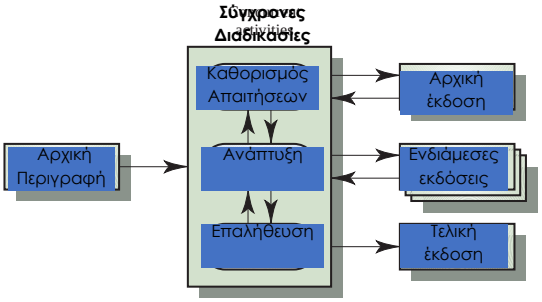
Ghezzi

V- Model

Το μοντέλο V δίνει έμφαση στο γεγονός ότι ο έλεγχος (επικύρωση και επαλήθευση) σχετίζεται με την ανάλυση και σχεδίαση και προκαλεί επαναλήψεις



Εξελικτικη Αναπτυξη



Αρχή: Do it Twice (Brooks 95)

Ghezzi

Εξελικτικη Αναπτυξη – 2 Εναλλακτικες

- Δημιουργία πρωτοτύπων
  - Στόχος είναι η συνεργασία με τον πελάτη και η δημιουργία ενός τελικού συστήματος ξεκινώντας από μία αρχική περιγραφή. Θα πρέπει να **ξεκινά με πολύ καλά κατανοητές απαιτήσεις**. Το σύστημα αναπτύσσεται προσθέτοντας σταδιακά νέα χαρακτηριστικά
  - Το αρχικό πρωτότυπο θα εξελιχθεί στο τελικό προϊόν
- Ανάπτυξη Throw-away
  - Στόχος είναι η κατανόηση των προδιαγραφών του συστήματος.. **Ξεκινά με τις λιγότερο κατανοητές απαιτήσεις**.
  - Το προϊόν πιθανότατα θα δημιουργηθεί με διαφορετικό τρόπο

Εξελικτικη Αναπτυξη

- Πλεονεκτήματα
  - Αντιμετωπίζει την ασάφεια στις απαιτήσεις
  - Παρέχει τη δυνατότητα στον πελάτη να αλλάξει γνώμη πριν υπογράψει
  - Μείωση χρόνου ανάπτυξης
  - Αρχικά πρωτότυπα χρησιμοποιούνται για εξοικείωση από τους χρήστες
  - Μεγαλύτερη πιθανότητα ανάπτυξης φιλικού προς το χρήστη λογισμικού
  - Ο πελάτης εμπλέκεται στην ανάπτυξη του προϊόντος
  - Αυξανόμενη σταδιακά ικανοποίηση του πελάτη
  - Επικοινωνία χρηστών / ομάδος ανάπτυξης

Εξελικτικη Αναπτυξη

- Προβλήματα
  - Έλλειψη παρατήρησης στη διαδικασία (αδυναμία πρόβλεψης επαναλήψεων)
  - Συστήματα λιτά δομημένα λόγω συχνών αλλαγών
  - Εστίαση στη λειτουργικότητα – Λιγότερη έμφαση στις μη λειτουργικές απαιτήσεις
  - Ειδικές ικανότητες μπορεί να απαιτηθούν (π.χ. γλώσσες για rapid prototyping)
  - Υπερβολικός ενθουσιασμός από τον πελάτη. Ενδεχόμενη υποεκτίμηση του χρόνου ανάπτυξης
  - Η δυνατότητα για συνεχείς τροποποιήσεις μειώνουν το βαθμό ικανοποίησης

Εξελικτικη Αναπτυξη

- Εφαρμογή
  - Για μικρά ή μεσαίου μεγέθους συστήματα (το αρχικό πρωτότυπο πλησιάζει το τελικό προϊόν)
  - Για τμήματα μεγάλων συστημάτων (π.χ.. the user interface ενός συστήματος εναέριας κυκλοφορίας)
  - Για συστήματα με μικρό χρόνο ζωής
  - Για συστήματα όπου υπάρχει αδυναμία έκφρασης των απαιτήσεων από πριν

  Ακατάλληλη για:
  - λογισμικό ενσωματωμένων συστημάτων
  - λογισμικό πραγματικού χρόνου
  - επιστημονικό λογισμικό

Τυπικοί Μετασχηματισμοί

- Οι προδιαγραφές (απαιτήσεις) του λογισμικού συντάσσονται με αυστηρά μαθηματικό τρόπο
- Η αρχική μαθηματική περιγραφή του λογισμικού μετασχηματίζεται σταδιακά στο τελικό προϊόν
- Διατήρηση της ορθότητας με χρήση μαθηματικών τεχνικών απόδειξης

Πλεονεκτήματα
- Σαφήνεια – Ακρίβεια
- Μεγάλη δυνατότητα αυτοματοποίησης

Μειονεκτήματα
- Η χρήση μαθηματικών συμβολισμών δυσχεραίνει την επικοινωνία με τον πελάτη
- Εκπαίδευση του προσωπικού στη χρήση => χρόνος, χρήμα

Διαχείριση Κίνδυνων – Risk Management

- Ενδεχομένως η κυριότερη αποστολή ενός manager είναι η ελαχιστοποίηση του ρίσκου
- Η έμφυτη έννοια του ρίσκου σε μία δραστηριότητα είναι ένα μέτρο της αβεβαιότητας ως προς το αποτέλεσμα
- Δραστηριότητες υψηλού ρίσκου αυξάνουν το κόστος και προκαλούν καθυστερήσεις
- Το ρίσκο σχετίζεται με την ποσότητα και την ποιότητα της διαθέσιμης πληροφορίας. Όσο λιγότερη πληροφορία τόσο μεγαλύτερο το ρίσκο.

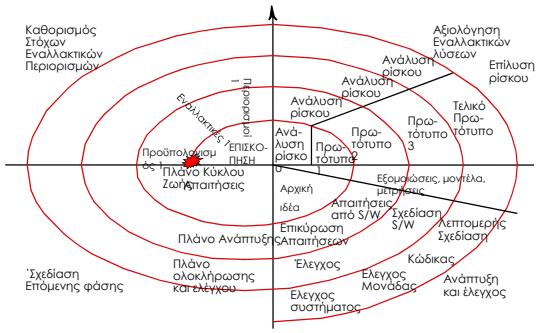
Διαχείριση Κίνδυνων – Risk Management

- Καταρράκτη
  - Υψηλό ρίσκο για νέα συστήματα λόγω προβλημάτων στις προδιαγραφές και στη σχεδίαση
  - Χαμηλό ρίσκο για πλήρως κατανοητά συστήματα που αναπτύσσονται με γνωστή τεχνολογία
- Δημιουργία πρωτοτύπων
  - Χαμηλό ρίσκο για νέες εφαρμογές λόγω του γεγονότος ότι οι προδιαγραφές και η υλοποίηση εξετάζονται ανά βήμα
  - Υψηλό ρίσκο λόγω έλλειψης παρατήρησης
- Τυπικοί Μετασχηματισμοί
  - Υψηλό ρίσκο λόγω απαιτήσεων για προηγμένη τεχνολογία και εξειδίκευση του προσωπικού

Παραδείγματα ρίσκου

- **Κατά τη διάρκεια ανάπτυξης εισαγωγή λογισμικού από ανταγωνιστή**
- **Αποχώρηση ατόμων από την ομάδα ανάπτυξης**
- **Ανικανότητα εμπρόθεσμης υλοποίησης ορισμένων σταδίων**
- **Το λογισμικό πληροί τις προδιαγραφές αλλά έχει πολύ μεγάλο χρόνο εκτέλεσης**
- **Μεγάλη κατανάλωση μνήμης**
- **Εμφάνιση νέων εργαλείων ανάπτυξης**
- **Λάθος κατανόηση μιας απαίτησης**
- **Αλλαγή ορισμένων από τις απαιτήσεις**
- **Αλλαγή του υλικού**
- **Μείωση προϋπολογισμού**
- **Βλάβες στο υλικό της ομάδας ανάπτυξης**

Σπειροειδες Μοντελο (Boehm 1988)



Σπειροειδές Μοντέλο - Παράδειγμα

Στόχος: Βελτίωση της ποιότητας του λογισμικού

Περιορισμοί: Χρονοδιάγραμμα 3 έτη
Χωρίς μεγάλες επενδύσεις

Εναλλακτικές: Επαναχρησιμοποίηση λογισμικού
Εισαγωγή τυπικών μεθόδων
Περισσότερος έλεγχος και εργαλεία ελέγχου

Ρίσκα: Απώλεια χρόνου χωρίς εμφανή αποτελέσματα
Η αύξηση της ποιότητας πιθανόν να διογκώσει το κόστος
Νέες τεχνικές ίσως διώξουν προσωπικό

Σπειροειδές Μοντέλο - Παράδειγμα

Επίλυση ρίσκου: **Επισκόπηση βιβλιογραφίας**
Πιλοτικό project
Αναζήτηση reusable components
Αξιολόγηση διαθέσιμων εργαλείων
Εκπαίδευση προσωπικού
Αποτελέσματα: Εμπειρία σε τυπικές μεθόδους περιορισμένη
Δυσκολία στην αξιολόγηση της βελτίωσης
Περιορισμένα εργαλεία
Reusable components αλλά όχι εργαλεία
Πλάνο: **Διερεύνηση τεχνικών επαναχρησιμοποίησης**
Διερεύνηση με χρήση πιλοτικού project
Συμφωνία: **Χρηματοδότηση για άλλους 12 μήνες**

Case Study: The Microsoft Synchronize and Stabilize Process

- Φιλοσοφία: Αποσύνθεση έργων λογισμικού σε πολλές μικρές ομάδες που εργάζονται παράλληλα, ωστόσο συμπεριφέρονται ως μία μεγάλη ομάδα ανάπτυξης
- Στόχος: Πλεονεκτήματα γρήγορης και ελεγχόμενης ανάπτυξης γνωστών μοντέλων, και ταυτοχρόνως ελευθερία και αυτονομία στους προγραμματιστές εντός των ομάδων
- Cusumano and Selby 1995: Synchronize and Stabilize

Planning Phase → Vision Statement (στόχοι, προτεραιότητες)
→ Specification document (αρχιτεκτονική)
→ Πρόγραμμα, προθεσμίες, ομάδες

30% των αρχικών απαιτήσεων τροποποιείται

Ghezzi

Case Study: The Microsoft Synchronize and Stabilize Process

- Κάθε ομάδα αποτελείται από δύο σύνολα ατόμων: Developers και Testers
- Συνεχής έλεγχος, Παράλληλη ανάπτυξη εντός ομάδος και μεταξύ ομάδων
- Συγχρονισμός:
  - Daily synchronization. Σε κάθε ομάδα τα ημερήσια αποτελέσματα εισάγονται σε βάση, ο πηγαίος κώδικας μεταγλωττίζεται και στη συνέχεια ελέγχεται
  - Product synchronization. Καθορισμός οροσήμων (milestones) όπου ελέγχεται αν το προϊόν έχει φθάσει σε σταθερή κατάσταση (συχνά ένα ορόσημο είναι precondition για διανομή του προϊόντος)
- Συνδυασμός hacker's culture (code and fix) με πειθαρχία και δομή στην ανάπτυξη λογισμικού

Ghezzi

Δυνατότητα Παρατήρησης (Visibility)

- Σε κάθε εταιρεία κάθε φάση στη σχεδίαση λογισμικού καταλήγει στη συγγραφή τεκμηρίωσης
- Ώστε η διοίκηση να έχει τη δυνατότητα παρατήρησης της προόδου (σε αντίθεση με άλλα έργα όπου ή πρόοδος είναι πιο εύκολα ορατή)

ΠΡΟΒΛΗΜΑΤΑ

- Οι manager ζητούν documents σε συγκεκριμένες ημερομηνίες (όταν οι designers πνίγονται στη δουλειά !!!)
- Η επικύρωση της τεκμηρίωσης απαιτεί επιπρόσθετο χρόνο
- Η υλοποίηση του προϊόντος δεν συμβαδίζει για πρακτικούς λόγους με το μοντέλο ανάπτυξης

Τεκμηρίωση – Παραγόμενα Έγγραφα

Δραστηριότητα	**Παραγόμενα Έγγραφα**
Ανάλυση Απαιτήσεων	Feasibility Study
Καθορισμός Απαιτήσεων	Requirements Specification
Σχεδίαση Συστήματος	Functional Specification
Σχεδίαση Λογισμικού	Design Specification
Ανάπτυξη	Program code
Έλεγχος Μονάδων	Unit test report
Έλεγχος Τμημάτων	Module test report
Έλεγχος Συστήματος	System test report
Παράδοση Προϊόντος	User Guide – Refer. Manual

Παρατηρησιμοτητα – Μοντέλα Ανάπτυξης

Μοντέλο Ανάπτυξης	**Δυνατότητα Παρατήρησης**
Καταρράκτη	Καλή. Έγγραφα ανά φάση
Εξελικτικό	Μειωμένη. Συχνές επαναλήψεις
Τυπικοί Μετασχ/μοί	Καλή. Παραγωγή εγγράφων δεδομένη
Επαναχρησιμοποίηση	Μέτρια
Σπειροειδές	Καλή. Έγγραφα ανά φάση

Επαγγελματικη Υπευθυνοτητα

Ο μηχανικός λογισμικού (software engineer) όπως κάθε επαγγελματίας θα πρέπει να τηρεί συγκεκριμένους νομικούς και ηθικούς κανόνες.

- Εχεμύθεια (σεβασμός πελάτη – εργοδότη)
- Υπευθυνότητα (Μη αποδοχή έργου πέραν των δυνατοτήτων)
- Πνευματικά Δικαιώματα
- Καλή Χρήση Υπολογιστών (no game-playing on employer's machine, no dissemination of viruses !!!)

Επαγγελματικη Υπευθυνοτητα

Ερώτημα:

Έστω ότι εργάζεστε σε εταιρεία που αναπτύσσει safety-critical λογισμικό για μηχανήματα έγχυσης χημειοθεραπευτικών.

και έστω ότι λόγω πίεσης χρόνου η εταιρεία δεν τηρεί τις προδιαγραφές ασφαλείας

Τι κάνετε ? Το δηλώνετε στον πελάτη (. . . . .και φεύγετε από την εταιρεία), τηρείτε εμπιστευτικότητα (. . . .και τίθεται σε κίνδυνο η ζωή ανθρώπων)

Αντικειμενοστραφής Ανάπτυξη Λογισμικού

Σε τι διαφέρει από τη συμβατική ανάπτυξη έργων λογισμικού ?

- **ΕΥΕΛΙΞΙΑ**: Η αντικειμενοστρεφής ανάπτυξη προσφέρει κατά τεκμήριο πολύ μεγαλύτερη ευκολία συντήρησης λογισμικού. Δηλαδή:
  - Διόρθωσης Σφαλμάτων (Corrective maintenance)
  - Τροποποίησης Απαιτήσεων (Adaptive maintenance)
  - Βελτίωσης Λογισμικού (perfective maintenance)
- Δομημένη Ανάπτυξη: Εάν οι αρχές της αντικειμενοστρεφούς ανάλυσης και σχεδίασης γίνουν κτήμα της ομάδας ανάπτυξης, η υλοποίηση λογισμικού γίνεται "ρουτίνα".
- Επαναληπτική ανάπτυξη: Στη βάση της αντικειμενοστρεφούς ανάλυσης και σχεδίασης είναι έμφυτη η έννοια της επανάληψης (επιστροφής σε προηγούμενες φάσεις)

ερωτήσεις απορίες

- Οι διαφάνειες βασίζονται στο http://users.uom.gr/~achat/AdvSoftEng/Intro_Software_Engineering.ppt